RADAMES JULIANO HALMEMAN

PROJETO DO COMPONENTE GERENCIADOR DE EXECUÇÃO DE WORKFLOW SEGUNDO A ABORDAGEM DE LINHA DE PRODUTO DE SOFTWARE

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática pelo Curso de Pós-Graduação em Informática, do Setor de Ciências Exatas da Universidade Federal do Paraná, em convênio com o Departamento de Informática da Universidade Estadual de Maringá.

Orientadora: Prof.ª Dr.ª Itana Maria de Souza Gimenes

CURITIBA

2003

Ministério da Educação
Universidade Federal do Paraná
Mestrado em Informática

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, do aluno Radames Juliano Halmeman, avaliamos o trabalho intitulado, "Projeto do Componente Gerenciador de Execução de Workflow Segundo a Abordagem de Linha de Produto de Software ", cuja defesa foi realizada no dia 22 de agosto de 2003, às dez horas, no Anfiteatro A do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação do candidato. (Convênio número 279-00/UFPR de Pós-Graduação entre a UFPR e a UEM - ref. UEM número 1331/2000-UEM).

Curitiba, 22 de agosto de 2003.

Profª. Dra. Itana Maria de Souza Gimenes
DIN/UEM (Orientadora)

Prof. Dr. Douglas Paulo Bertrand Renaux
CPGEI/CEFET-PR– Membro Externo

Prof. Dr. Marcos Sfair Sunyé
DINF/UFPR – Membro Interno

AGRADECIMENTOS

Agradeço a Deus pela constante presença em minha vida.

A minha esposa, Cristina, pelo apoio, paciência e renúncia durante este importante período.

A toda a minha família pelo constante incentivo.

Aos amigos Edson Alves de Oliveira Junior pela constante ajuda e apoio e ao Sr. Edson Alves de Oliveira pela hospitalidade.

Ao amigo Fabrício Ricardo Lazilha, pelo auxílio nos trabalhos, pela solidariedade e incentivo nos momentos de desânimo e dificuldades.

Agradeço à equipe de trabalho do Laboratório de Engenharia de Software da UEM, especialmente aos integrantes do Projeto ExPSEE.

Agradeço aos amigos José Hilário Delconte Ferreira pelo auxílio nos momentos de dificuldades e correria, aos amigos Sérgio Neres Domingos e Sandra Regina Postali que estiveram sempre presentes auxiliando e incentivando.

E finalmente, mas não menos importante, a Itana, pela orientação relativa não somente ao trabalho desenvolvido como também pela amizade, ajuda e compreensão das minhas limitações e dificuldades.

SUMÁRIO

LISTA DE ABREVIATURAS

CORBA ..... *Common Object Request Broker Architecture*

DBC ..... Desenvolvimento Baseado em Componentes

ExPSEE ..... *Experimental Process-centered Software Engineering Environment*

GUI ..... *Graphical User Interface*

OCL ..... *Object Constraint Language*

OMG ..... *Object Management Group*

RUP ..... *Rational Unified Process*

UML ..... *Unified Modeling Language*

WAPI ..... *Workflow Application Programming Interface*

WfMC ..... *Workflow Management Coalition*

WfMS ..... *Workflow Management System*

XML ..... *Extensible Markup Language*

LISTA DE FIGURAS

RESUMO

A engenharia de *software* busca constantemente por um conjunto de processos, técnicas e ferramentas que propiciem o desenvolvimento de produtos com qualidade e que sejam economicamente viáveis.

A reutilização é uma das técnicas deste conjunto. Considera-se que ao se reutilizar partes bem especificadas, desenvolvidas e testadas pode-se construir *software* em menor tempo e com maior confiabilidade. Há um número, sempre crescente, de técnicas e propostas de técnicas que favorecem a reutilização. Entre elas estão a engenharia de domínio, *frameworks*, padrões, arquitetura de *software* e desenvolvimento baseado em componentes. No entanto, neste contexto falta uma maneira sistemática e previsível para realizar a reutilização. A abordagem de linha de produto de *software* preenche esta lacuna, pois, tem como principal objetivo possibilitar a reutilização de maneira sistemática e previsível, não abolindo as demais técnicas, mas considerando-as como complementares.

A abordagem de linha de produto é aplicável a sistemas que compartilham um conjunto gerenciado de características, que satisfazem necessidades específicas de um segmento ou missão e que são desenvolvidos a partir de um núcleo de artefatos seguindo um plano previamente definido. Deste modo, percebe-se que o domínio dos Sistemas Gerenciadores de *Workflow* é propício à aplicação desta abordagem.

A tecnologia de *workflow* tem apresentado um significante crescimento nos últimos anos o que implica na necessidade de novas técnicas de engenharia de *software* para facilitar construção deste tipo de sistema.

Esta dissertação apresenta o projeto do componente Gerenciador de Execução de *Workflow (WorkflowExecutionMgr)* segundo a abordagem de linha de produto de *software*. O componente *WorkflowExecutionMgr* se caracteriza por executar um *workflow* previamente instanciado através do gerenciamento de suas tarefas e foi projetado para permitir diferentes variantes de algoritmos de escalonamento possibilitando a instanciação de produtos com características diferentes.

O projeto do componente seguiu um processo e uma arquitetura de linha de produto para Sistemas Gerenciadores de *Workflow* previamente definidos. A validação do componente proposto foi realizada através da implementação de um protótipo. As contribuições deste trabalho incluem o projeto do componente Gerenciador de Execução de *Workflow* que incrementa o núcleo de artefatos para a arquitetura de linha de produto para WfMS e a revisão da arquitetura previamente proposta.

Palavras-Chaves: Linha de Produto; Workflow; Componentes; Arquitetura de Software; Processo.

ABSTRACT

The software engineering area has been constantly looking for processes, techniques and tools that enable the development of high quality products at economically feasible costs.

Reuse is amongst these techniques. It is considered that the reuse of parts well specified, developed and tested, increases the reliability of software products as well as allowing rapid development. There has been an increasing number of techniques that encourages software reuse, such as domain engineering, frameworks, patterns, software architecture and component based development. However, it seems that we are still missing a systematic and predictable means to effectively apply software reuse. The software product line approach can be viewed as a way of filling this gap. The objective of this approach is to allow software reuse based on well-defined processes, artefacts and rules. It encompasses most of the reuse techniques previously defined.

The software product line approach is applicable to systems that share a manageable set of characteristics that fulfils specific needs of a sector or mission (domain). It considers products that can be developed from a core set of artefacts following a well-defined production plan. Taking this into account, the Workflow Management Systems domain is a potential candidate for the application of this approach. The use of these systems have been significantly increasing during last years, thus efficient software engineering techniques that facilitates the development of these systems are required.

This dissertation presents the design of the component Workflow Execution Manager (WorkflowExecutionMgr) according to the software product line approach. The Component WorkflowExecutionMgr manages the task execution of a previously instantiated workflow. It was designed to allow different scheduling algorithms so that products with different characteristics can be instantiated.

The component design followed both a software product line architecture and a development process previously defined. A prototype was developed in order to validate the component design. The contributions of this work include the component design that increments the artefacts core set of the product line as well as the revision of the software architecture previously defined.

Keywords: Product line; Workflow; Components; Software Architecture; Process.

CAPÍTULO 1 - INTRODUÇÃO

A engenharia de software busca constantemente por um conjunto de processos, técnicas e ferramentas que propiciem o desenvolvimento de produtos com qualidade e que sejam economicamente viáveis. A reutilização é uma das técnicas deste conjunto. Considera-se que ao se reutilizar partes bem especificadas, desenvolvidas e testadas pode-se construir software em menor tempo e com maior confiabilidade.

A abordagem de linha de produto de *software* está entre as técnicas que tem como objetivo promover a reutilização no processo de *software*. Uma linha de produto de *software* é uma coleção de produtos de *software* que compartilham um mesmo conjunto gerenciado de características que satisfazem necessidades específicas de um segmento ou missão.

Esta abordagem é aplicável a domínios em que se faz necessária à construção de sistemas similares mas com algumas características diferentes.

Neste sentido, percebe-se que os Sistemas Gerenciadores de *Workflow* (WfMS – *Workflow Management Systems*) são favoráveis à aplicação da abordagem de linha de produto de software. *Workflow* é a automatização total ou parcial do processo de negócio de uma organização. Um processo de negócio é o conjunto de procedimentos e atividades que conduzem a realização do objetivo de negócio de uma organização. WfMS são utilizados para definir, criar e gerenciar *workflows*. Eles executam uma ou mais máquinas de *workflow* capazes de interpretar uma definição de processo, interagir com os participantes do *workflow* e invocar o uso de ferramentas ou aplicativos externos.

Nos últimos anos, muitas organizações têm se preocupado com a melhoria da qualidade de seus processos, pois isso implica em aumento da qualidade e redução de custos de produção. Deste modo, essa organização tem se interessado pela tecnologia de *workflow*.

Para o desenvolvimento de WfMS, organizações como a WfMC (*Workflow Management Coalition)* (WfMC, 2001) têm propostas de arquiteturas genéricas que apresentam o conjunto básico de funcionalidades necessárias para os produtos de *workflow*.

Seguindo a abordagem de linha de produto de *software*, LAZILHA (2002) especificou uma arquitetura de linha de produto para WfMS a partir da arquitetura genérica e do modelo de referência da WfMC (1995). Este trabalho se concentrou na concepção da arquitetura, embora os componentes tenham sido descritos e alguns aspectos de variabilidade analisados, nenhum componente foi desenvolvido.

O desenvolvimento dos componentes para essa arquitetura é importante tanto para povoar a arquitetura, complementando assim o núcleo de artefatos de linha de produto, quanto para validar a arquitetura proposta.

Dando prosseguimento ao trabalho desenvolvido por LAZILHA (2002) , esta dissertação apresenta o projeto do componente *WorkflowExecutionMgr*, que faz parte dessa arquitetura. A principal funcionalidade deste componente é a execução de um *workflow* previamente instanciado através do gerenciamento e execução de suas tarefas.

O desenvolvimento do componente seguiu o processo proposto por LAZILHA (2002) que se baseia em um método para desenvolvimento baseado em componentes com a adição de alguns estereótipos para a representar variabilidade. Como ferramenta de apoio utilizou-se o Rational Rose (RATIONAL, 2002).

A validação do componente foi realizada através da implementação de um protótipo. Este protótipo permitiu a validação das funcionalidades do componente e de suas interfaces, bem como o refinamento do projeto.

As principais contribuições do trabalho estão no incremento do núcleo de artefatos, na revisão e no refinamento da arquitetura para WfMS proposta por LAZILHA (2002).

Esta dissertação está organizada da seguinte maneira. No Capítulo 2 estão os conceitos da abordagem de linha de produto de *software*, do Desenvolvimento Baseado em Componentes (DBC) e do método Catalysis (D'DOUZA; WILLS, 1999). No Capítulo 3 são apresentados os conceitos da tecnologia de *workflow*, o projeto ExPSEE (GIMENES *et al.*, 2001) e a arquitetura de linha de produto de *software* para WfMS proposta por LAZILHA (2002) que constitui o contexto para o componente proposto. No Capítulo 4 está o Projeto do Componente Gerenciador de Execução de *Workflow* e, finalmente, no Capítulo 5 estão as Conclusões e as propostas de Trabalhos Futuros.

CAPÍTULO 2 - A ABORDAGEM DE LINHA DE PRODUTO DE SOFTWARE

Este Capítulo apresenta uma introdução à abordagem de linha de produto de *software*. Na Seção 2.1, são apresentados à definição e os conceitos básicos. Na Seção 2.2 é ressaltada a importância da representação das variabilidades. Na Seção 2.3, são apresentadas as atividades essenciais para o desenvolvimento da uma linha de produto de *software*. Na Seção 2.4, estão os conceitos de Desenvolvimento Baseado em Componentes, pois neste trabalho foi realizado o projeto de um componente utilizando esta técnica. Na Seção 2.5, é feita uma breve introdução ao método Catalysis que foi utilizado no desenvolvimento do projeto do componente *WorkflowExecutionMgr*. Finalmente, na Seção 2.6, são citadas algumas abordagens para linha de produto de *software* e descrito o método Kobra, que apresenta várias similaridades com o processo utilizado neste trabalho.

2.1. Definição e Conceitos Básicos

Segundo CLEMENTS (2001) uma linha de produto de *software* é definida como uma coleção de sistemas que compartilham um mesmo conjunto gerenciado de características que satisfazem necessidades específicas de um segmento ou missão e que são desenvolvidos a partir de um núcleo de artefatos seguindo um plano previamente definido.

A abordagem de linha de produto de *software* visa a redução de custos e incremento da qualidade dos produtos. Isto é alcançado principalmente por meio da reutilização dos artefatos de *software* produzidos. Um artefato de *software* é um item reutilizável de *software* utilizado como bloco de construção de uma linha de produto.

Ainda existem controvérsias sobre o enfoque de linha de produto ao considerarmos técnicas anteriores com propósitos semelhantes, principalmente a engenharia de domínio e *frameworks*. Porém, deve-se considerar essas técnicas como complementares. Deste modo, o enfoque de linha de produto de *software* pode ser visto como uma forma de organização dessas técnicas para facilitar o gerenciamento de famílias de produtos (GIMENES; TRAVASSOS, 2002).

A abordagem de linha de produto de *software* é aplicável aos domínios em que há necessidade de produtos específicos, mas com um conjunto de semelhanças e pontos de variabilidade bem definidos, formando assim uma família de produtos (LAZILHA, 2002).

Para se obter sucesso em uma linha de produto de *software* é necessário um gerenciamento sistemático e cuidadoso planejamento das variabilidades enquanto se explora os pontos semelhantes. As semelhanças garantem a reutilização dos artefatos de *software* e o gerenciamento das variabilidades permitirá que produtos com características diferentes sejam criados a partir de uma mesma arquitetura. Pela importância do conceito de variabilidade ele será destacado na próxima Seção.

2.2. Variabilidade na Linha de Produto de Software

Os produtos de uma linha de produto de *software* compartilham um conjunto de características. Contudo, para que se torne possível à reutilização dos artefatos produzidos em diferentes produtos, as variações entre os produtos devem ser identificadas e representadas.

Segundo GIMENES e TRAVASSOS (2002), variações são diferenças tangíveis que podem ser reveladas e distribuídas entre os artefatos da linha de produto, sejam eles a arquitetura, os componentes, as interfaces entre os componentes ou as conexões entre componentes. As variações podem ser identificadas em qualquer fase do ciclo de desenvolvimento de uma linha de produto. A expressão de uma variação pode ser obtida através da introdução de parâmetros instanciáveis em tempo de construção que associados aos componentes, subsistemas ou coleção de subsistemas permitem a configuração do produto (CLEMENTS, 2001). De acordo com BACHMAN e BASS (2001), os projetistas normalmente preparam a arquitetura para mudança adicionando mecanismos para possibilitar essas mudanças. Contudo, existe uma carência na documentação sobre o tratamento de variação e de que maneira estas variações podem ser implementadas. Entende-se por tratamento de variação desde o momento do reconhecimento de um ponto de variação até o mapeamento do ponto para uma instância de variação (GIMENES; TRAVASSOS, 2002).

Em orientação a objetos, por exemplo, as variações podem ser representadas, em nível de projeto, por meio de especializações de determinadas classes.

Segundo BACHMAN e BASS (2001) é necessário representar diferentes possibilidades nos artefatos de uma arquitetura quando a representação está sendo gerada e não se sabe qual alternativa será escolhida no momento de se gerar um novo produto.

Para realizar a representação das variações deve-se considerar as causas de variação e os tipos de variação. Conforme discutido a seguir.

- ***Variação funcional:*** acontece quando uma função existe em alguns produtos, mas não existe em outros. Por exemplo, considere um carro com um dispositivo que seja rádio e sistema de navegação. Podemos ter um carro apenas com rádio, outro apenas com sistema de navegação e outro com ambos.
- ***Variação nos dados:*** a estrutura dos dados pode variar de um produto para outro. Por exemplo, se considerarmos uma aplicação em que dois componentes compartilham informações sobre clientes. As informações dos clientes possuem, entre outras coisas, o endereço do cliente armazenado como uma cadeia de caracteres não estruturada. Para permitir que uma característica diferente seja implementada em uma nova versão (mostrar o endereço de forma estruturada, por exemplo) então o formato para o endereço do cliente deverá ser diferente.
- ***Variação no fluxo de controle:*** um padrão de interação pode variar de um produto para outro. Por exemplo, considere um mecanismo de notificação entre componentes, o qual informe aos componentes interessados que houve mudanças em alguns valores de dados. Uma possível solução para informar os demais componentes seria que todos outros componentes fossem notificados em seqüência dentro de um fluxo de controle. No entanto, se considerarmos um ambiente distribuído, pode ser que alguns dos componentes não sejam notificados, o que causa um comportamento imprevisível do sistema. Para resolver este problema, poderia ser modificado

o fluxo de controle de tal modo que os componentes enviassem um código de controle informando se recebeu a notificação.

- ***Variação na tecnologia:*** A tecnologia pode variar sob muitos aspectos. Se considerarmos a plataforma que é composta do sistema operacional, *hardware*, *middleware*, interfaces com o usuário entre outras coisas. Percebe-se que um desses elementos pode variar, do mesmo modo que a variação funcional. Por exemplo, um determinado elemento de *middleware* pode ser necessário em um produto, mas não em outro.
- ***Variação nas metas de qualidade:*** as metas de qualidade podem variar de um produto para outro. Por exemplo, se considerarmos a forma como é feita a comunicação entre fabricantes e clientes acerca de dificuldades na utilização de algum produto. Esta comunicação pode ser realizada por meio de um mecanismo que envia mensagem ao cliente (não on-line) ou através de uma conexão permanente, a última opção oferece auxilio imediato ao cliente. A escolha de uma destas formas de comunicação implica em variação nas metas de qualidade, que podem variar de um produto para outro.
- ***Variação no ambiente:*** a maneira como um produto interage com o ambiente pode variar. Por exemplo, um determinado *middleware* pode ser invocado por diferentes linguagens.

Independente da causa de uma variação ela sempre estará enquadrada em um tipo de variação; os tipos estão descritos abaixo:

- ***Opcional:*** a variação é opcional se, por exemplo, a funcionalidade está em um produto, mas não em outro.
- ***Instância de várias alternativas:*** a arquitetura permite que várias funcionalidades alternativas sejam inseridas. Por exemplo, um automóvel disponibiliza um conector para onde pode ser adicionado um dispositivo de controle de viagem. Esse dispositivo pode ser de alta tecnologia e alto custo para automóveis submetidos a condições extremas ou de baixa tecnologia e baixo custo para automóveis para uso em condições normais.

- ***Conjunto de instâncias para várias alternativas:*** a arquitetura fornece a possibilidade de várias instâncias nas quais múltiplas alternativas podem ser inseridas. Por exemplo, um produto de *software* capaz de se comunicar com o mundo externo utilizando protocolos de comunicação diferentes em paralelo. Esse produto é composto de um conjunto de protocolos de comunicação enquanto outro produto pode ter outro conjunto de protocolos. Para cada um dos protocolos, pode-se ainda, considerar diferentes funcionalidades.

Depois de identificadas as causas e o tipo da variação é possível promover a representação destas variações. Um estudo mais detalhado de como realizar a representação pode ser encontrado em (BACHMAN; BASS, 2001).

Outra maneira para representar os pontos de variação é através do conceito de *features*. Segundo GRISS (1998) uma *feature* é uma característica de um produto que clientes e usuários vêem como importante na descrição e distinção dos produtos que integram uma linha de produto de *software*. Uma *feature* pode ser um requisito específico ou uma seleção entre os requisitos opcionais e alternativos.

Uma *feature* pode estar relacionada às características do produto, tais como: funcionalidade, usabilidade e desempenho ou relacionada às características de implementação, tais como o tamanho, a plataforma de execução, ou a compatibilidade com certos padrões.

Um modelo de *features* é composto pela representação dos tipos de *features* que podem ocorrer em uma família de produtos e seus inter-relacionamentos. O modelo geralmente é representado por um grafo em que os nós contêm as *features* e seus atributos. O grafo contém ainda decorações para indicar *features* opcionais, obrigatórias e composição de *features*.

O modelo de *features* e o modelo de casos de uso apresentam alguma relação, mas algumas diferenças devem ser observadas conforme GRISS (1998) *apud* GIMENES e TRAVASSOS (2002). São elas:

- o modelo de *features* é orientado para o reutilizador e o modelo de casos de uso é orientado para o usuário;

- o modelo de casos de uso descreve os requisitos do usuário em termos de funcionalidades do sistema enquanto o modelo de *features* organiza o resultado da análise dos aspectos comuns e variáveis com o objetivo de preparar a base para a reutilização;
- o modelo de casos de uso deve cobrir todos os requisitos de um sistema individual do domínio. Já no modelo de *features*, devem estar representadas as características que o analista do domínio considera importantes, pois este resume apenas os itens essenciais relativos aos objetivos do domínio;
- a notação utilizada para representar casos de uso é diferente daquela utilizada para representar *features*, pois nem todas as *features* podem ser automaticamente relacionadas como casos de uso;
- mesmo existindo um grupo de *features* básicas que o usuário vê como capacidades do sistema, nem todas essas *features* aparecem nos casos de uso. Algumas *features* surgem: no detalhamento da implementação, nas opções de configuração do sistema, ou por sugestão de especialistas do domínio. Tais *features* podem aparecer somente nas fases de projeto e implementação.

Uma discussão sobre features relacionadas com casos de uso é feita por JACOBSON, GRISS e JONSSON (1997). Uma feature é definida como um caso de uso ou parte deste. Neste contexto é sugerida a utilização do estereótipo <<extend>> para representar aspectos de variação em casos de uso. O caso de uso estendido recebe uma marca, representada por um círculo preenchido para indicar um ponto de variação, como mostrado na Figura 2.1.

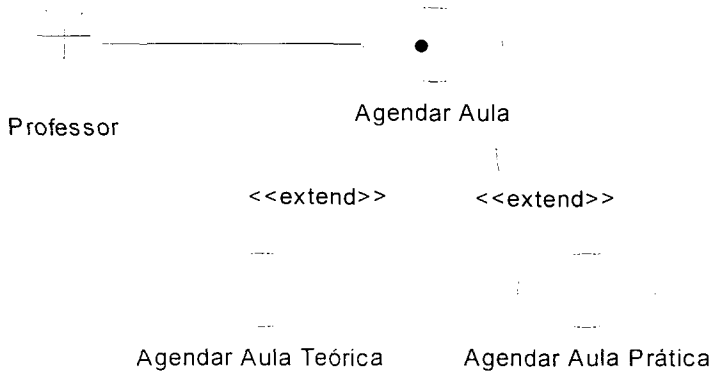

Figura 2.1.: Exemplo de representação de variabilidade em casos de uso (LAZILHA, 2002).

A próxima seção apresenta as atividades essenciais para o desenvolvimento de uma linha de produto de *software*.

2.3. Atividades Essenciais para o Desenvolvimento de Linha de Produto

De acordo com CLEMENTS (2001), o desenvolvimento de uma linha de produto envolve três atividades essenciais: Desenvolvimento do Núcleo de Artefatos, Desenvolvimento de Produtos e Gerenciamento. Estas atividades são mostradas na Figura 2.2.

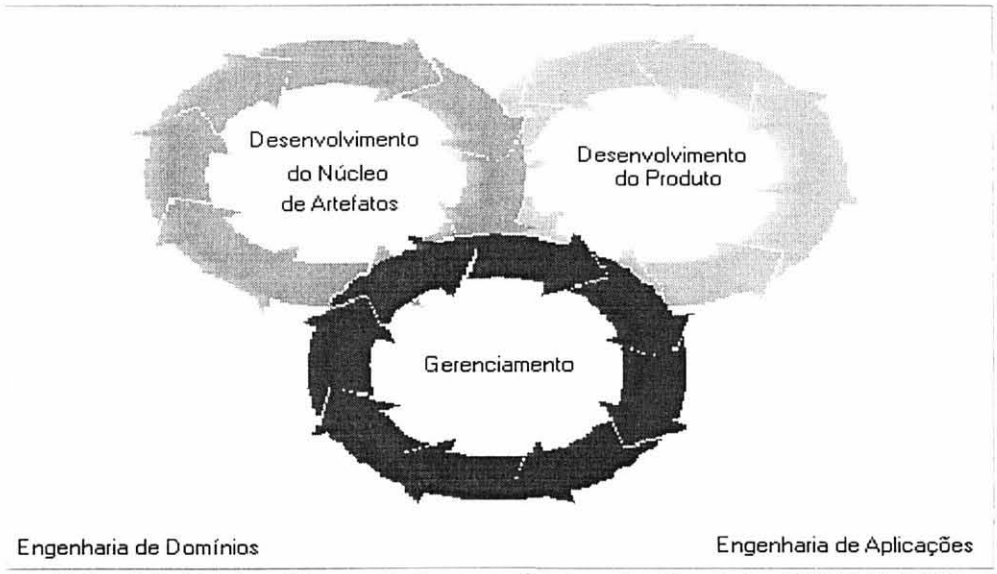

Figura 2.2.: Atividades Essenciais para Linha de Produto (CLEMENTS, 2001).

Na Figura 2.2, cada círculo representa uma atividade essencial e as três estão em constante movimento e dependem umas das outras. Essencialmente são desenvolvidos o Núcleo de Artefatos e o Produto sob o amparo do Gerenciamento.

A literatura apresenta algumas atividades com nomes diferentes, CLEMENTS E NORTHROP (2001), reconhecem explicitamente que a atividade de desenvolvimento do núcleo de artefatos tem sido chamada de engenharia de domínio e que a atividade de desenvolvimento do produto tem sido chamada de engenharia da aplicação. WEISS (1999) reconhece que família de produtos também é conhecida pelo termo domínio.

O processo de desenvolvimento de uma linha de produto, de acordo com GIMENES e TRAVASSOS (2002), pode ser visto como dois modelos de ciclo de

vida, Desenvolvimento do Núcleo de Artefatos e Desenvolvimento do Produto, conforme destacados na Figura 2.3.

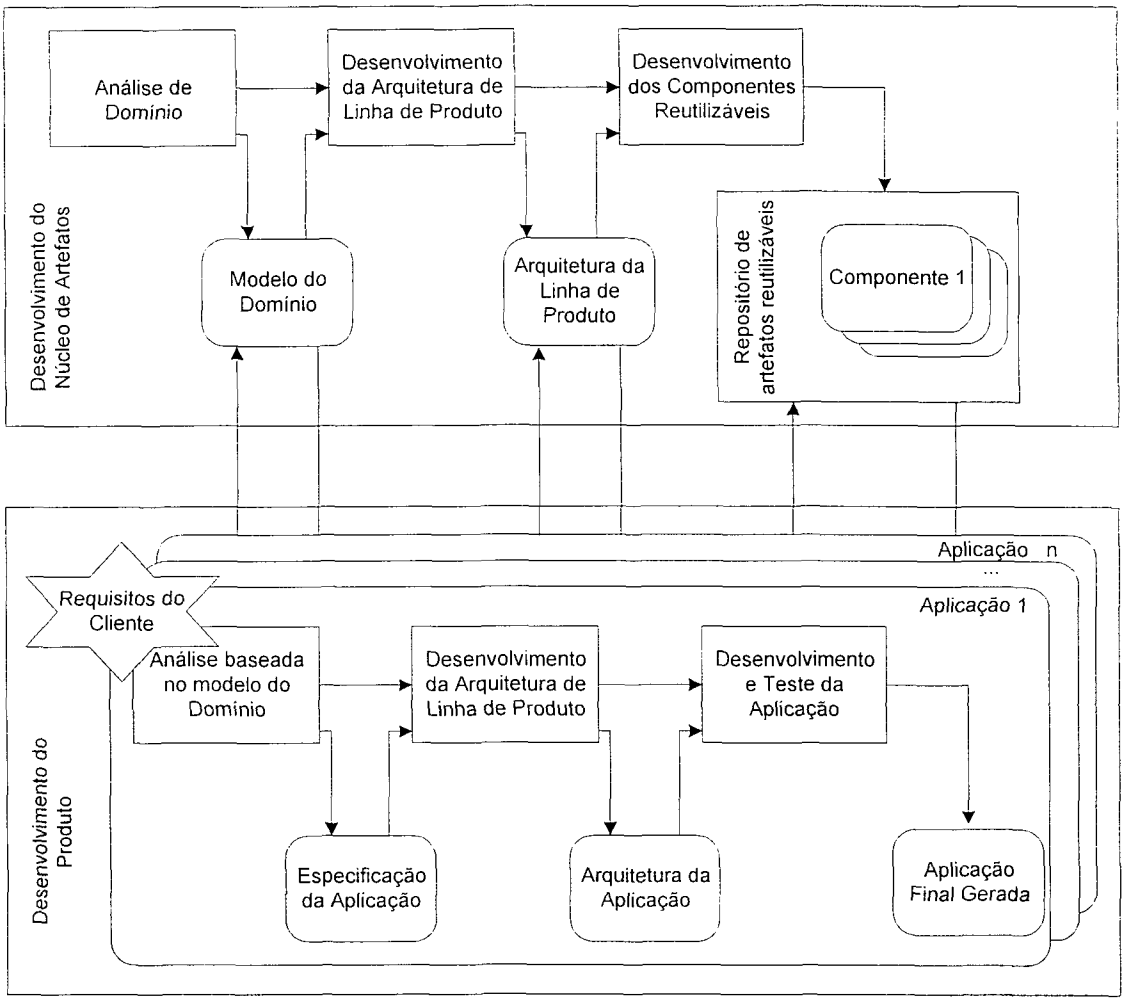

Figura 2.3: Processo de Desenvolvimento de uma Linha de Produto (GIMENES;TRAVASSOS, 2002).

Nesta figura os retângulos representam as etapas de desenvolvimento enquanto os retângulos arredondados representam os artefatos produzidos. A a estrela apresenta os requisitos da aplicação a ser desenvolvida (produto). O modelo de desenvolvimento do núcleo de artefatos é composto de três etapas: análise de domínio, desenvolvimento da arquitetura e desenvolvimento de componentes reutilizáveis. Estas produzem um modelo de domínio, uma arquitetura e um conjunto de componentes reutilizáveis e geradores de *software* para a linha de produto.

O desenvolvimento do produto, é composto das etapas: análise baseada no modelo de domínio, desenvolvimento da arquitetura de linha de produto e

desenvolvimento e teste da aplicação. Estas produzem a especificação da aplicação, arquitetura da aplicação e a aplicação final gerada.

O desenvolvimento do produto, também é conhecido como engenharia de aplicação, *em geral, inclui* os seguintes artefatos:

- o modelo do domínio, base para identificar os requisitos do cliente;
- um framework de arquitetura de linha de produtos, base para especificar uma arquitetura para um membro da família;
- um conjunto de componentes reutilizáveis a partir do qual um subconjunto de componentes será integrado à arquitetura para gerar um produto.

2.3.1. Desenvolvimento do Núcleo de Artefatos

A idéia básica do desenvolvimento do núcleo de artefatos é disponibilizar um conjunto de artefatos que permita a construção de produtos membros de uma família. Como resultado desta atividade teremos o domínio da linha de produto (definição do contexto), o núcleo de artefatos e o plano de produção.

2.3.1.1. Domínio da Linha de Produto

O domínio da linha de produto é a descrição dos produtos que constituirão a linha de produto ou quais produtos a linha será capaz de incluir. O domínio da linha de *produto deve ser* cuidadosamente definido, uma definição muito ampla pode comprometer a identificação das variações.

2.3.1.2. Núcleo de artefatos

O núcleo de artefatos, representado na Figura 2.3 como Repositório de artefatos reutilizáveis, é a base para a construção de produtos em uma linha de produto. É composto da arquitetura e do conjunto de componentes de software que são desenvolvidos para reutilização na linha de produto. Inclui também modelos de desempenho, resultados da avaliação da arquitetura, documentação de projeto,

especificação de requisitos, modelos de domínio e componentes COTS(*Commercial off-the-shelf*).

2.3.1.3. Plano de Produção

O plano de produção descreve como os produtos são produzidos a partir do núcleo de artefatos. Em uma linha de produto o plano de produção é o que direciona a reutilização. O plano de produção descreve os requisitos de variação entre os produtos, como e quais ferramentas específicas serão utilizadas, qual a ordem de utilização das ferramentas e a maneira de promover a junção dos artefatos produzidos.

O plano de produção gerará as seguintes entradas para a atividade de desenvolvimento do núcleo de artefatos:

- ***Restrições do produto:*** descreve os elementos em comum e as variações existentes entre os produtos que irão constituir a linha de produto, as características de comportamento, os padrões que devem ser seguidos, os limites de desempenho que devem ser observados, os sistemas com os quais o produto deverá interagir, as restrições de *hardware*, os requisitos de qualidade que devem ser observados, as características de mercado e as novas tecnologias que poderão beneficiar a linha de produto no futuro;
- ***Estilos de arquiteturas, frameworks e padrões:*** estes elementos são utilizados para projetar as entradas necessárias para o núcleo de artefatos;
- ***Restrições de produção:*** determinam quais os padrões específicos das organizações deverão ser aplicados no desenvolvimento da linha de produto;
- ***Estratégia de produção:*** abrange tudo que se refere à concretização do núcleo de artefatos;
- ***Repositório de artefatos já existentes:*** o repositório inclui todos os possíveis artefatos já existentes em sistemas legados e através de uma análise cuidadosa é possível identificar quais serão os mais adequados para compor o núcleo de artefatos.

2.3.2. Desenvolvimento do Produto

A atividade de desenvolvimento do produto, também é conhecida como engenharia da aplicação, depende dos seguintes elementos:

- modelo do domínio da linha de produto;
- núcleo de artefatos, especificamente um *framework* de arquitetura de linha de produto que é utilizado como base para especificar uma arquitetura para um membro da família;
- conjunto de componentes reutilizáveis a partir do qual um subconjunto de componentes será integrado à arquitetura para gerar o produto;
- plano de produção;
- requisitos específicos para os produtos individuais.

A Figura 2.4, mostra este relacionamento. As setas duplas demonstram a forte relação existente entre os diversos artefatos.

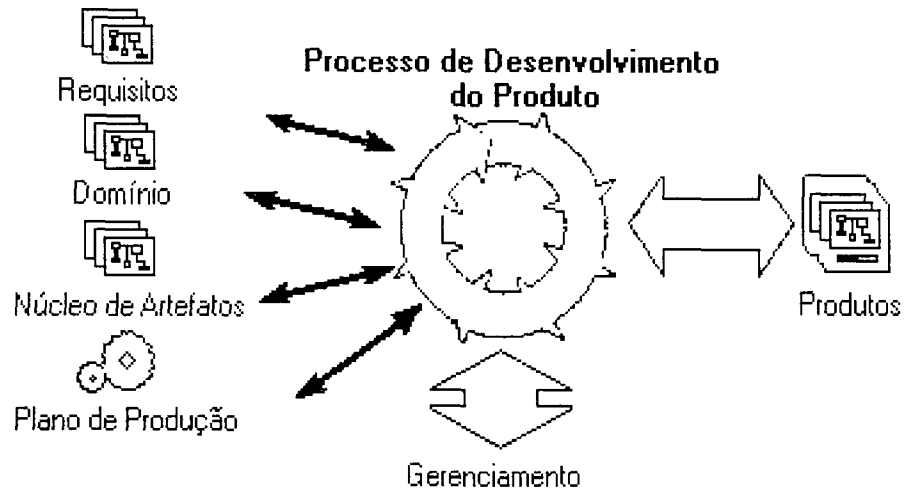

Figura 2.4: Desenvolvimento do Produto (SEI, 2002).

2.3.3. Gerenciamento

Gerenciamento é extremamente importante para o desenvolvimento de uma linha de produto. Para as atividades deve-se oferecer artefatos, coordenação e supervisão.

Para a construção de uma linha de produto, um forte gerenciamento deve ser exercido, tanto no nível técnico quanto no nível organizacional. O gerenciamento é importante no período de desenvolvimento e também para manter a integridade da linha de produto.

O gerenciamento técnico incide diretamente sobre o desenvolvimento do núcleo de artefatos e no desenvolvimento de produtos. Ele deve garantir que estes elementos estejam dentro dos requisitos da linha de produto, de acordo com os planos de produção e coletar dados para a verificar o progresso das atividades.

O gerenciamento organizacional está voltado para a estrutura da organização, deve verificar e garantir que os produtos e recursos produzidos estejam de acordo com as necessidades da organização.

As etapas de desenvolvimento do produto e Gerenciamento estão fora do escopo deste trabalho.

Na próxima seção serão apresentados os conceitos básicos do Desenvolvimento Baseado em Componentes.

2.4. Desenvolvimento Baseado em Componentes

O crescente amadurecimento dos métodos de DBC tem contribuído com a abordagem de linha de produto. Os componentes são os elementos que preenchem a arquitetura de uma linha de produto. Dessa forma, a disponibilidade de mecanismos para especificar e construir componentes de forma a permitir variações entre estes e que possam facilitar a geração de aplicações com características diferentes com menor esforço de desenvolvimento é essencial para o enfoque de linha de produto.

Segundo Werner (2000), o desenvolvimento de grande parte dos produtos de *software* disponíveis no mercado está baseado em uma abordagem de desenvolvimento em blocos monolíticos, formados por um grande número de partes inter-relacionadas, estando esses relacionamentos, na maioria das vezes, implícitos.

O DBC surgiu como uma nova perspectiva para o desenvolvimento de *software*, cujos objetivos incluem: quebrar esses blocos monolíticos em componentes interoperáveis, reduzir a complexidade no desenvolvimento, reduzir os custos de desenvolvimento através da utilização de componentes que podem ser adequados a outras aplicações.

Segundo D'SOUZA e WILL (1999), um componente pode ser definido como uma unidade de *software* independente, que encapsula dentro de si seu projeto e implementação, e oferece interfaces bem definidas para o meio externo, para que este componente possa se unir a outros componentes e dar origem aos sistemas baseados em componentes. As interfaces são chamadas de Interfaces Fornecidas ou Interfaces Requisitadas. A interface fornecida define as operações que o componente oferece a outros componentes. A interface requisitada define as operações que o componente requisitará de outros componentes. As interfaces servem apenas para a comunicação entre componentes ocultando dos usuários os detalhes de implementação. A especificação de um componente é, normalmente, publicada separadamente de seu código fonte por meio da especificação das interfaces oferecidas por ele (GIMENES et al., 2000).

Segundo BOSCH (2000), componentes podem ser classificados em diferentes tipos:

- componentes desenvolvidos pelo próprio cliente,
- componentes adquiridos para um determinado domínio e
- componentes COTS que são componentes genéricos de prateleira encontrados no mercado. Durante o desenvolvimento de um produto a partir da arquitetura da linha de produto, esses três tipos de componentes podem ser selecionados para preencher a arquitetura do produto específico.

A técnica de desenvolvimento baseado em componentes tem por objetivo fornecer um conjunto de ferramentas, técnicas e notações que possibilitem que, ao longo do processo de *software*, ocorra tanto a produção de novos componentes quanto à reutilização de componentes existentes. O amadurecimento dessas técnicas contribui muito com a abordagem de linha de produto, oferecendo mecanismos para especificar e construir componentes de forma a permitir variações entre estes e que possam facilitar a geração de aplicações com características

diferentes com menor esforço de desenvolvimento. Estes métodos estão focados nos seguintes elementos: componentes, interfaces e construção de componentes. A notação utilizada é a *Unified Modeling Language* (UML) (BOOCH; RUMBAUGH; JACOBSON, 1999). Alguns exemplos destes métodos são o Processo Unificado da Rational (RUP – *Rational Unified Process*) (KRUCHTEN, 2000) e o método Catalysis, que foi escolhido para guiar o processo de desenvolvimento do componente *WorkflowExecutionMgr* projetado neste trabalho. O método Catalysis será descrito na próxima seção.

2.5. O Método Catalysis

O método Catalysis incorpora conceitos importantes que apóiam o desenvolvimento de *software* para sistemas baseados em objetos e componentes. O método baseia-se na linguagem de modelagem UML e alinha-se aos padrões de sistemas de objetos distribuídos abertos, construídos a partir de componentes e *frameworks*.

Catalysis oferece modelos precisos e um processo de desenvolvimento completo e sistemático, permitindo aos desenvolvedores partirem da análise e especificação do domínio da aplicação e chegarem ao código, particionando o sistema e identificando ao longo do processo os elementos de reutilização.

O método baseia-se em princípios de abstração, precisão e compatibilização de partes. Esses princípios podem ser aplicados em todos os níveis de desenvolvimento de *software* a partir do domínio de um problema. A abstração propõe a visualização clara dos aspectos essenciais do problema, tais como o que o sistema deve fazer, a definição da arquitetura e da concorrência entre funções. A precisão é importante para evitar a ambigüidade de compreensão e para a análise de correspondência entre níveis de abstrações diferentes. A compatibilização de partes está intimamente ligada a reusabilidade, com o objetivo de reaproveitar os componentes, *frameworks*, padrões e especificações que constituem o *software*.

Existe uma grande preocupação em relação à interface dos componentes. Esta abordagem sugere que a colaboração entre os componentes para a formação da arquitetura seja feita através das interfaces. Por isso, existe todo um formalismo para a definição das interfaces, com especificação de restrições, detalhamento da

interface, entre outros, de forma que esta consistência possa ser verificada de maneira formal.

Neste trabalho, adotou-se o mesmo processo utilizado por (LAZILHA, 2002). Este processo envolve o uso do método Catalysis (que é um método de propósito geral com ênfase no DBC) e seus estágios (especificação de requisitos, especificação do sistema, projeto da arquitetura e projeto interno dos componentes) com algumas modificações para a representação das variabilidades.

Na próxima seção serão citadas algumas abordagens existentes de linha de produto. Uma descrição mais detalhada será feita sobre o método Kobra, pois este método possui similaridades com o processo adotado por (LAZILHA, 2002) no desenvolvimento da arquitetura de linha de produto de *software* para WfMS, e conseqüentemente no processo utilizado neste trabalho.

2.6. Abordagens Existentes de Linha de Produto

Esta Seção cita algumas abordagens que suportam o enfoque de linha de produto de software. Segundo LAZILHA (2002), algumas abordagens são abrangentes ao apresentar soluções para questões relacionadas à representação de aspectos ligados à engenharia de domínio e aplicação, assim como relacionados aos conceitos referentes à evolução, gerenciamento, análise da relação de custos e benefícios, tomadas de decisão orientada ao mercado e avaliação de riscos. Outras fornecem métodos concentrados em alguns dos aspectos pertencentes à tecnologia de linha de produto, tais como a definição da família de produtos, construção da infra-estrutura arquitetural básica ou mesmo a implementação de componentes reutilizáveis. Neste caso, elas são usualmente integradas a algum outro método específico de engenharia de *software* como, por exemplo, análise de *features*, desenvolvimento baseado em componentes ou utilização da notação UML ao longo de um processo de desenvolvimento de *software*.

Algumas abordagens existentes são a *Synthesis* que foi documentada pelo *Software Productivity Consortium* (SPC, 2003), *Family-Oriented Abstraction, Specification and Translation* (FAST) (WEISS; CHI TAU, 1999), *Product Line Software Engineering* (PuLSE) do Centro de *Fraunhofer* (BAYER, J. et al.,1999)

Feature-Oriented Domain Analysis (FODA) documentado pelo SEI (KANG, 1990), a iniciativa PLP (*Product Line Practice*) do SEI (SEI, 2002) e a abordagem proposta por (BOSCH, 2000). uma descrição detalhada destas abordagens é apresentada em (GIMENES;TRAVASSOS, 2002). Essas abordagens são baseadas em engenharia de domínio. Por esse motivo, são menos eficientes na representação de arquiteturas e componentes. Assim, os métodos de DBC podem ser usados no processo de desenvolvimento de linha de produto para reduzir o intervalo entre a análise do domínio, arquitetura, projeto interno dos componentes e implementação. Iniciativas similares ao processo proposto por LAZILHA (2002) já podem ser encontradas na literatura, tais como o Kobra (ATKINSON; BAYER; MUTHING, 2000). Devido a esta similaridade, a seguir uma breve caracterização do método Kobra será apresentada.

2.7. O Método Kobra

O método Kobra (ATKINSON; BAYER; MUTHING, 2000) integra os conceitos de linha de produto e desenvolvimento baseado em componentes. Os princípios de construção e utilização de linhas de produto são baseados na metodologia PuISE (BAYER, J. et al., 1999). O método KobrA considera o desenvolvimento e manutenção de um *framework* o que inclui todas as alternativas possíveis de realização da arquitetura de produtos membros. O conceito central do método é um componente KobrA (Komponent) descrito em diferentes níveis de abstração e organizados em forma de uma árvore. Cada Komponent é descrito em dois níveis de abstração, sendo o primeiro, uma especificação que define as propriedades externas visíveis de um Komponent e comportamentos, e uma realização descrevendo como um Komponent preenche suas responsabilidades quando relacionados a outros Komponents de mais baixo nível. A descrição da especificação e realização de um Komponent é realizada através da utilização de diagramas UML.

A especificação de um Komponent é derivada da estrutura arquitetural da linha de produto e fornece a definição da interface do Komponent utilizando quatro modelos básicos: estrutural, comportamental, funcional e decisão. Estes modelos representam respectivamente o estado do Komponent, sua reação a estímulos externos, seu impacto sobre entidades externas e a natureza de trocas no comportamento do Komponent dependendo do ambiente.

A realização do Komponent é também representada por quatro modelos principais: interação, estrutural, atividade e decisão. Estes modelos são relacionados aos modelos de especificação, mas ao contrário deles, representam como um Komponent realiza sua funcionalidade ao invés de mostrar o que é a funcionalidade. Se a interface especificada de um Komponent está de acordo com a interface fornecida por um componente pré-existente, como por exemplo, COTS ou algum sistema legado reestruturado, ele poderia ser utilizado normalmente com algum ajuste (GIMENES;TRAVASSOS, 2002).

O método KobrA está sendo atualmente avaliado num contexto industrial através do desenvolvimento de um estudo de caso para o domínio de Planejamento de Recursos Organizacionais.

2.8. Considerações Finais

Este Capítulo apresenta conceitos relevantes para o contexto deste trabalho. Apresentou os conceitos básicos para linha de produto de *software* e seus elementos, a importância e os conceitos para se representar as variabilidades e caracterizou as atividades essenciais para a produção de uma linha de produto de *software* (Desenvolvimento do Núcleo de Artefatos, o Desenvolvimento do Produto e o Gerenciamento), dessas atividades foi descrita em maiores detalhes o Desenvolvimento do Núcleo de Artefatos, pois o componente *WorkflowExecutionMgr* incrementar o repositório de componentes para a linha de produto. Foram conceituados o DBC, o Método Catalysis e as abordagens existentes de linha de produto de *software,* destacando o método Kobra pela sua similaridade com o processo utilizado neste trabalho.

No próximo Capítulo serão apresentados os conceitos para a Tecnologia de *Workflow*, o projeto ExPSEE, a proposta de arquitetura para linha de produto de *software* proposta por LAZILHA (2002) .

CAPÍTULO 3 - A TECNOLOGIA DE *WORKFLOW*

Este Capítulo, na Seção 3.1, apresenta uma introdução à tecnologia de *workflow* visando destacar os principais conceitos. Nesta dissertação foram adotados os padrões propostos pela WfMC. A WfMC é uma organização internacional que tem como objetivo promover a área de *workflow* divulgando a tecnologia e desenvolvendo padrões para a interoperabilidade de sistemas de *workflow*. Um dos elementos principais, para este trabalho, são os conceitos para os sistemas gerenciadores de *workflow*, apresentados na Seção 3.4.

Na Seção 3.5, será apresentado o projeto ExPSEE (*Experimental Process-centered Software Engineering Enviroment*), este ambiente de engenharia de *software* permite a cooperação no desenvolvimento e execução de processos de *software*. Por existir grande semelhança entre um processo de *software* e um *workflow* cujo negócio principal é a produção de *software*, muitos componentes de um ambiente de engenharia de *software* podem ser utilizados em sistemas de gerenciamento de *workflow*. Assim, o estudo do ambiente e seus componentes contribuem no projeto do componente *WorkflowExecutionMgr*.

A Seção 3.6, apresenta a arquitetura de linha de produto para *Workflow Management System*, proposta por (LAZILHA, 2002). O componente *WorkflowExecutionMgr* faz parte dessa arquitetura. Portanto, será apresentada a arquitetura e uma descrição das principais funcionalidades de cada um de seus componentes. O projeto do componente propiciou uma revisão da arquitetura, revelando detalhes não detectados durante o processo de concepção da arquitetura. Deste modo, algumas modificações foram realizadas.

3.1. Definição de *Workflow*

Um processo de negócio é o conjunto de procedimentos e atividades que conduzem à realização da meta de negócio de uma organização. A competitividade e a busca por maneiras mais eficientes para atingir seus objetivos fazem com que as organizações procurem novas tecnologias que as auxiliem na realização de suas metas. Deste modo, reengenharia e automação de processos são necessidades vitais para os empreendimentos contemporâneos. Neste contexto surge a tecnologia

de *workflow* que propõe a automação total ou parcial dos processos de negócio (WFMC, 2001).

Geralmente *workflow* é definido como a automação de um processo de negócio de forma total ou parcial, de tal forma que se possa, de uma maneira mais eficiente controlar, simular e maximizar a eficiência dos processos de negócio.
Segundo a WfMC, *workflow* é a automação total ou parcial de um processo de negócio, durante o qual documentos ou tarefas são passadas de um participante para outro, a fim de que sejam realizadas ações de acordo com uma coleção de regras e procedimentos (WFMC, 2001).

3.2. Conceitos e Termos Relacionados a *Workflow*

As relações da terminologia associada a um *workflow* pode ser observada na Figura 3. 1, e são descritos a seguir.

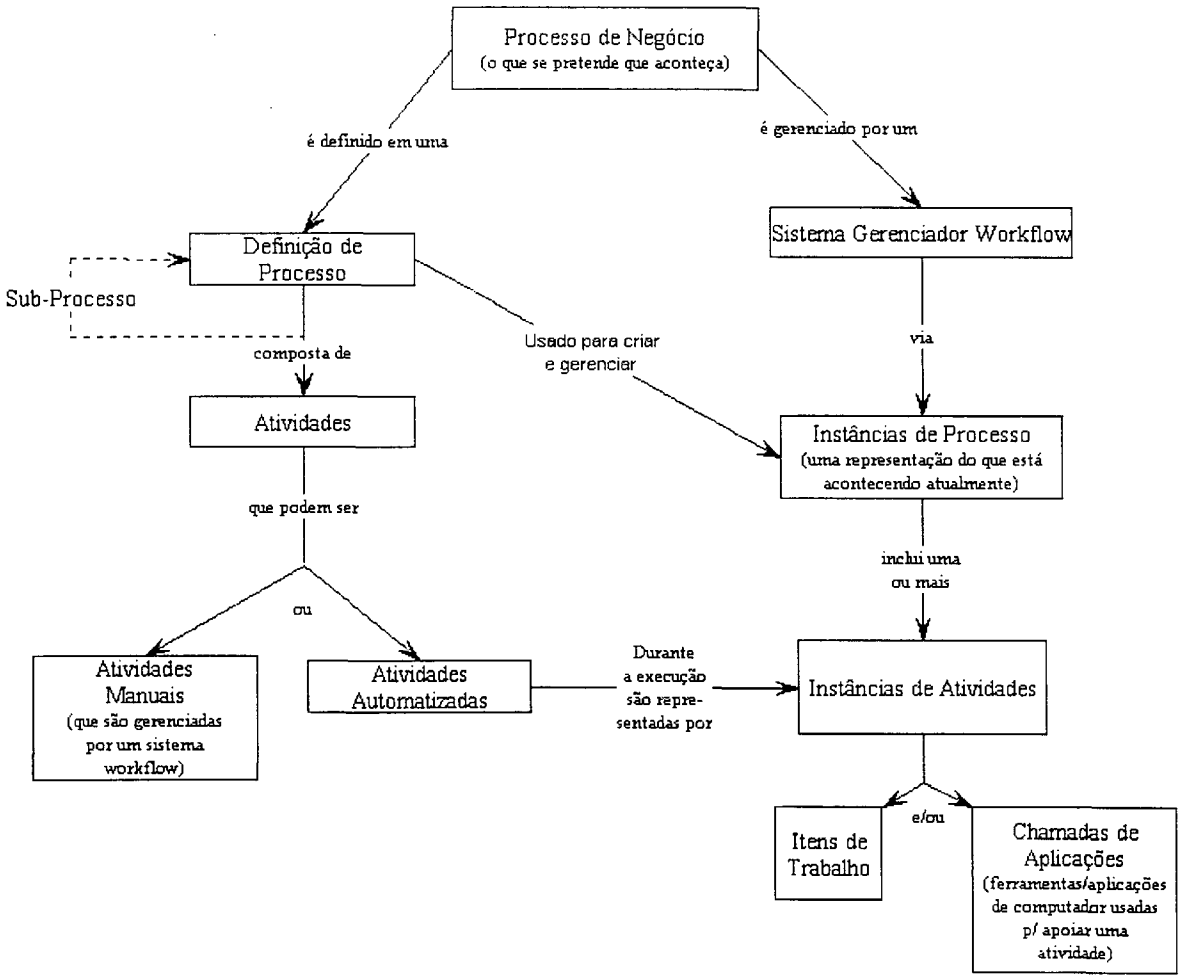

Figura 3. 1.: Relação da Terminologia Associada a um *Workflow* (WFMC, 1995).

- ***Processo de negócio***: segundo a WfMC, é um conjunto de um ou mais procedimentos ou atividades os quais coletivamente realizam um objetivo de negócio ou verificam o cumprimento de uma meta. Normalmente, dentro do contexto de uma estrutura organizacional, em que existem regras funcionais e de relacionamentos (WFMC, 2001).
- ***Sistema gerenciador de workflow (WfMS):*** os sistemas gerenciadores de *workflow* propiciam os meios para a automação dos processos de negócios, solicitando recursos humanos ou de máquinas de acordo com os procedimentos de determinada atividade e gerenciam as atividades de trabalho (WFMC, 1995).
- ***Definição de processo*:** é a representação de um processo de tal maneira que este permita manipulações automatizadas, como a modelagem ou execução por um sistema de gerenciamento de *workflow*.
- ***Sub-processo:*** diz-se de um processo quando este é chamado ou executado a partir de outro processo (ou sub-processo) e que faz parte do processo global.
- ***Tarefa*:** o mesmo que atividade.
- ***Atividade:*** é a descrição de uma parte ou porção de trabalho que forma um passo lógico dentro de um processo. Uma atividade pode ser *manual* (aquela que não permite automação computadorizada) ou um *workflow* (automatizada). Uma atividade de *workflow* requer recursos humanos ou materiais que permitam a execução do processo. Uma atividade é alocada para um participante do *workflow* (WFMC, 1999).
- ***Instância de processo:*** é a representação de uma única execução de um processo. Esta instância pode ser criada, gerenciada ou terminada por um WfMS, de acordo com a definição de processo.
- ***Instância de atividade:*** é a representação de uma atividade em uma única execução de um processo. Esta instância pode ser criada e gerenciada por um WfMS, de acordo com a definição de processo (WFMC, 1995).
- ***Item de trabalho:*** é uma partição de uma atividade. Deste modo, pode-se dizer que as atividades são representadas como um conjunto de itens de trabalho.

- ***Chamada de aplicação:*** é uma aplicação de *workflow* que é invocada pelo WfMS para automatizar, totalmente ou parcialmente, uma atividade, ou auxiliar um participante do *workflow* no processamento de um item de trabalho.
- ***Participante:*** o mesmo que ator de um *workflow*.
- ***Ator:*** é responsável pela execução total ou parcial de uma determinada instância de atividade.
- ***Cargo/função (papel):*** é um atributo de participante que determina suas responsabilidades e direitos na execução de um *workflow*.

3.3. Arquitetura Genérica para *Workflow*

Baseada em comparações entre os produtos disponíveis no mercado, a WfMC identificou a viabilidade da construção de um modelo de implementação genérico de um sistema de *workflow* que pode ser ajustado à maioria dos produtos do mercado LAZILHA (2002). Este modelo identifica os principais componentes funcionais e suas respectivas interfaces dentro de um sistema de gerenciamento de *workflow*. A arquitetura genérica está representada na Figura 3. 2.

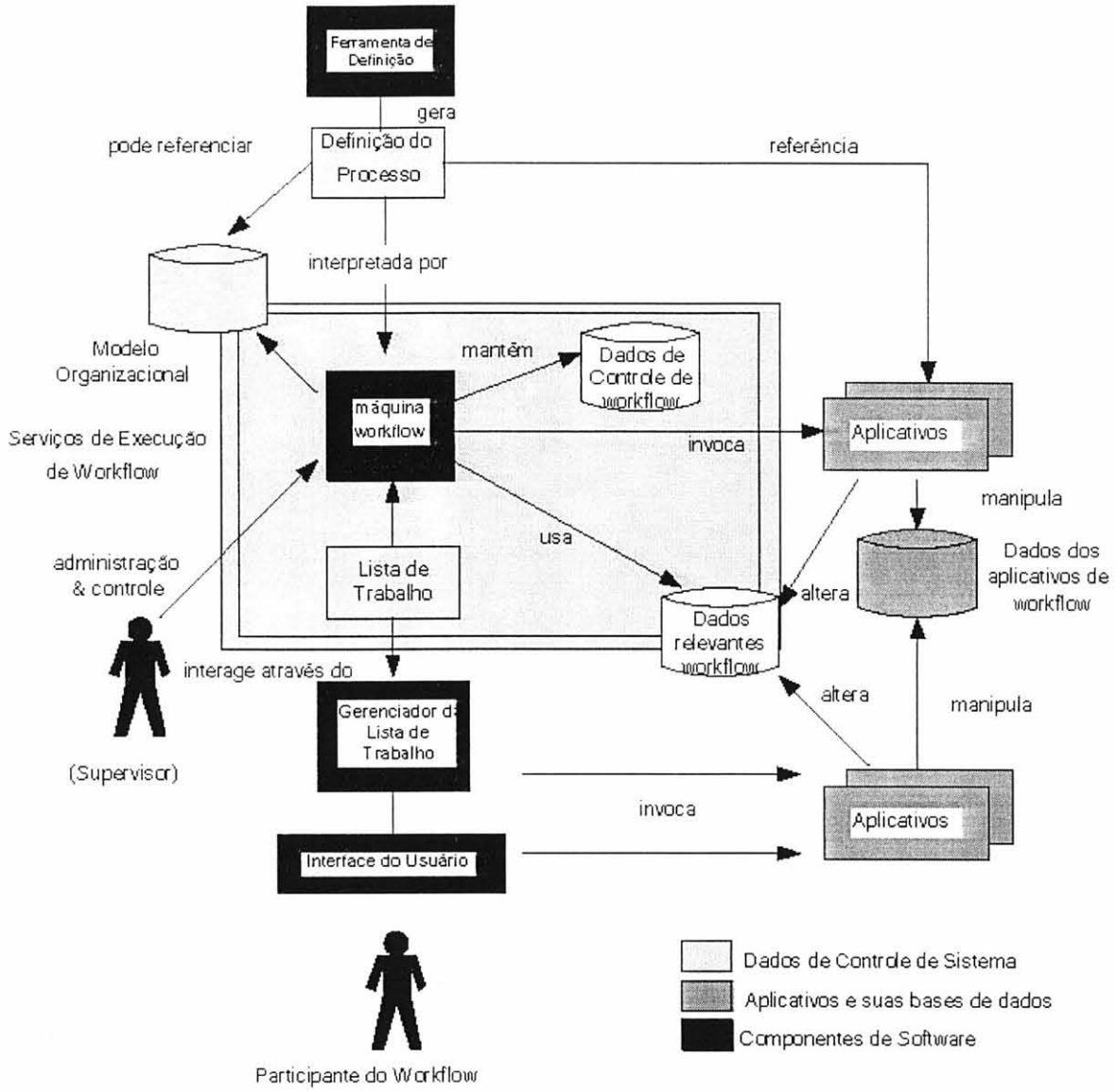

Figura 3. 2.: Arquitetura Genérica de um Sistema de Workflow (WFMC, 1995).

A partir deste modelo, cada implementação de um sistema gerenciador de *workflow* pode optar por adaptar certas estruturas (componentes ou interfaces) de acordo com as necessidades da aplicação. A arquitetura genérica é composta por três tipos de componentes (WFMC, 1995):

- ***Componentes de software*:** são os componentes que oferecem suporte as funcionalidades do WfMS (como ferramenta de definição, máquina de *workflow*, gerenciador da lista de trabalho e interface do usuário).

- ***Dados de controle do sistema***: são dados utilizados pelos componentes de *software* (dados de controle de *workflow*, dados relevantes do *workflow* e o modelo organizacional).
- ***Aplicativos e suas bases de dados***: são os aplicativos que não fazem parte dos WfMSs, mas podem ser invocados por ele como parte do sistema de *workflow*;

3.4. Sistemas Gerenciadores de *Workflow* (WfMS)

Sistemas gerenciadores de *workflow,* de acordo com a WfMC, são os sistemas que definem, criam e gerenciam a execução de *workflows* através do uso de produtos de *software*, executando uma ou mais máquinas *workflow*, os quais estão aptos a interpretar a definição de um processo, interagir com os participantes do *workflow* e, onde necessário, invocar o uso de ferramentas da tecnologia da informação ou aplicativos (WFMC, 1995).

Os WfMS propiciam os meios para a automação dos processos de negócios, de acordo com os procedimentos de determinada atividade, solicitam recursos humanos ou de máquinas e gerenciam atividades de trabalho (WFMC, 1995).

Segundo a WfMC (1995), estes gerenciadores fornecem suporte a três áreas funcionais:

- ***Construção do Workflow***: é a definição do processo de *workflow* e suas tarefas. É o mapeamento formal do processo de negócio do mundo real para que este possa ser entendido por um computador.
- ***Interação entre Participantes***: Tratam da interação com usuários humanos e aplicações para o processamento de várias etapas de uma atividade.
- ***Execução do Workflow***: encarrega-se do gerenciamento do *workflow* em um ambiente operacional e do sequenciamento das várias atividades a serem tratadas em cada processo.

A Figura 3.3 mostra a relação entre as áreas funcionais.

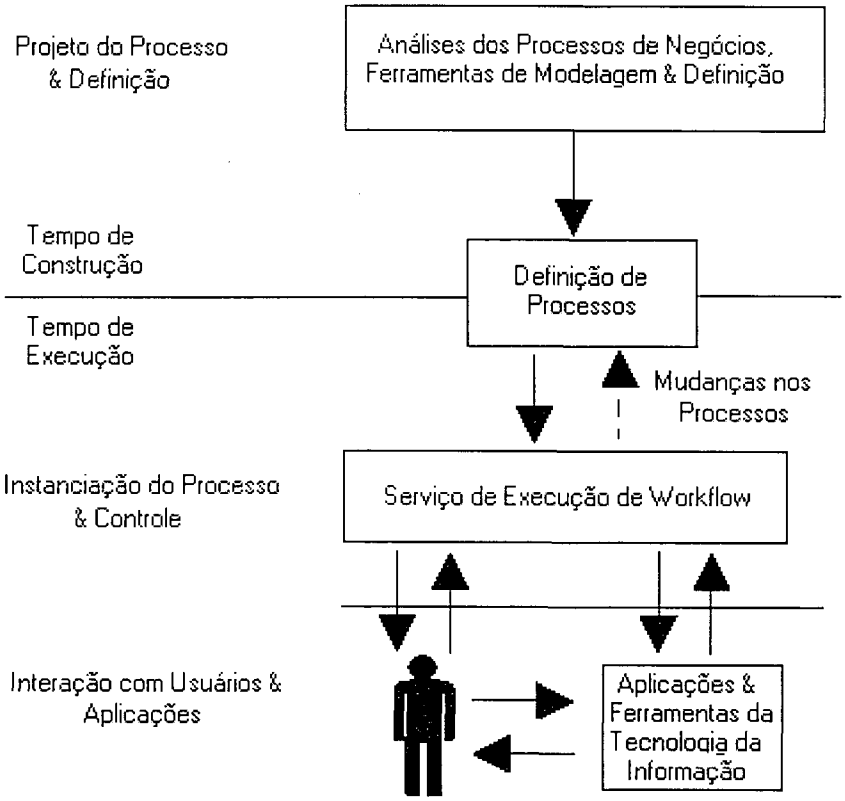

Figura 3.3.:Relação entre as Áreas Funcionais (WFMC, 1995).

3.4.1. Modelo de Referência para *Workflow* Proposto pela WfMC

O Modelo de Referência para WfMS, é a proposta da WfMC para padronizar o desenvolvimento de aplicações baseadas na tecnologia *workflow*, e propõe uma arquitetura básica para o desenvolvimento das aplicações.

A Figura 3.4 ilustra a maioria dos componentes e interfaces que compõem arquitetura de *workflow*.

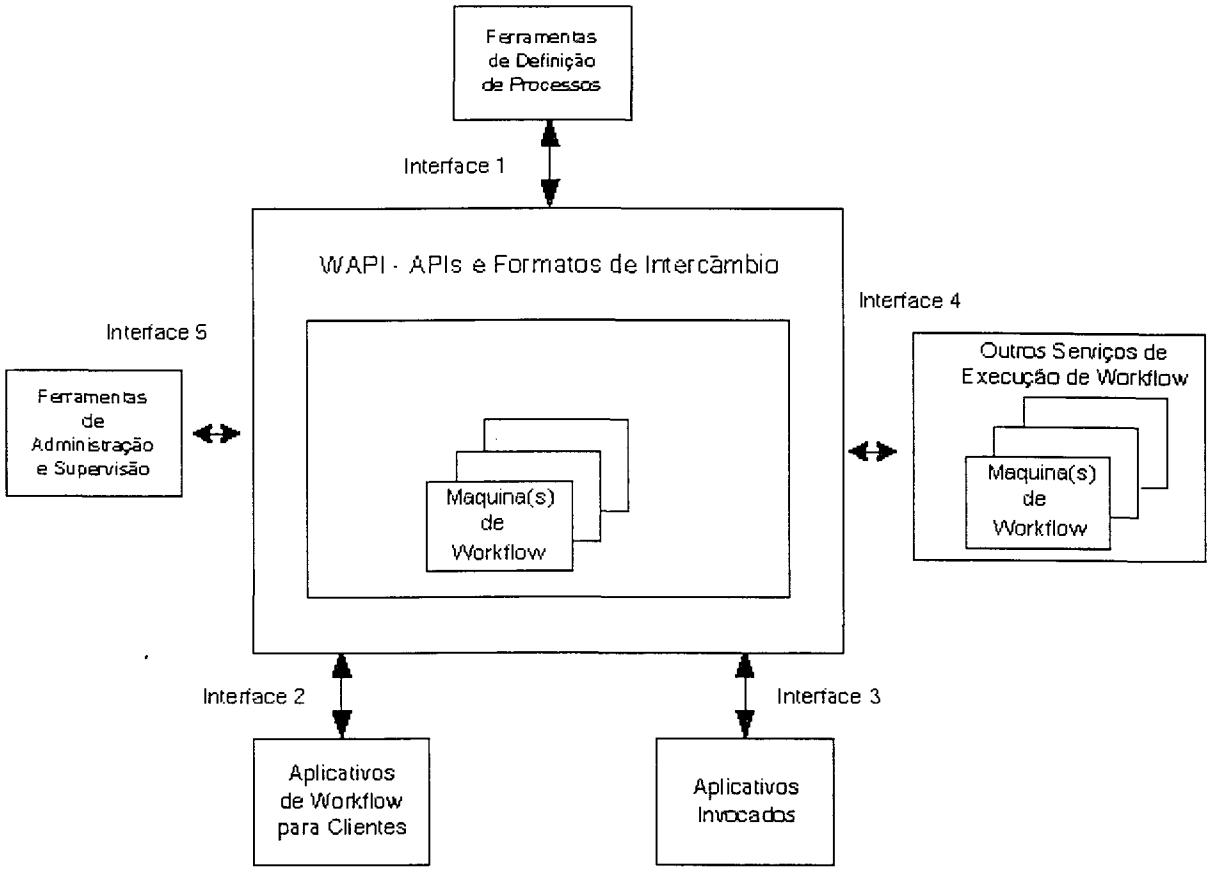

Figura 3.4.: *Workflow Reference Model* - Componentes & Interfaces (WFMC, 1995).

A seguir estão as descrições dos componentes do Modelo de Referência (WFMC, 1995).

- ***Serviço de execução do workflow***: serviço de *software* que consiste de uma ou mais máquinas de *workflow* para criar, gerenciar e executar instâncias de *workflow*. As aplicações comunicam-se com este serviço através de WAPI (*Workflow Application Programing Interface*). Este serviço fornece o ambiente e os mecanismos de execução nos quais as ativações e instanciações dos processos acontecem. A interação com o serviço de execução ocorrerá via interface de aplicações de *workflow* para clientes (Interface 2) ou interface de aplicações invocadas (Interface 3).
- ***Definição de processos***: Diferentes ferramentas podem ser utilizadas para analisar, modelar, descrever e documentar um processo de negócio (Interface 1). Pressupõe-se que cada produto que trabalhe com *workflow* forneça uma ferramenta para este fim. É importante que ao final do projeto e modelagem o

processo de definição possa ser interpretado e executado pelas máquinas *workflow*.

- ***Aplicações de workflow para clientes*:** é o *software* gerenciador de listas de trabalhos que interage com o usuário final nas atividades que envolvem recursos humanos (Interface 2). Uma lista de trabalho é uma lista de itens de trabalho atribuídos a um usuário ou grupo de usuários por uma máquina *workflow*.
- ***Aplicações invocadas*:** o WfMS necessita invocar determinada aplicação com o objetivo de automatizar uma atividade, de forma total ou parcial, que faça parte de um item de trabalho de um participante (Interface 3). Estas invocações podem ser feitas via chamada local simples ou chamada de um componente remoto (objeto distribuído). Existe a necessidade de mecanismos de integração que garantam a comunicação entre a máquina de *workflow* e a aplicação invocada.
- ***Interoperabilidade entre serviços de execução de workflow*:** permite a troca de itens de trabalho entre diferentes WfMSs (Interface 4). Logo, diferentes máquinas *workflow* podem trabalhar em conjunto. O foco de padronização neste item está na forma de cooperação entre os WfMS que pode ser estabelecida entre os WfMSs.
- ***Ferramentas de supervisão e administração*:** estas ferramentas de supervisão e administração permitem a avaliação do estado geral e extração de métricas do sistema, o que é importante para as organizações (Interface 5).

3.5. O Projeto ExPSEE

Desenvolvido desde 1994, pelo grupo de Engenharia de *Software* da Universidade Estadual de Maringá, o ambiente ExPSEE (*Experimental Process-centered Software Engineering Enviroment*) (GIMENES *et al.*, 2001) é um ambiente de Engenharia de *Software* que permite a cooperação no desenvolvimento e execução de processos de *software*. O objetivo deste ambiente é oferecer uma plataforma experimental para alunos de graduação e pós-graduação, assim como

investigar novas tecnologias sobre processos de *software* e mecanismos para sua modelagem e automação.

A correlação dos ambientes de Engenharia de *Software* e Sistemas Gerenciadores de *Workflow* foi realizada inicialmente por TANAKA (2000), através de comparação entre estes ambientes. Para tal comparação foram utilizados o modelo de processos do *Process-Manager* produzido no ExPSEE, o modelo de referência e a arquitetura Genérica para WfMS da *Workflow Management Coalition*. Com base nestes estudos também se produziu um *Framework* de Agenda de Tarefas, que é uma parte deste ambiente. Por existir grande semelhança entre estes modelos, um processo de *software* pode ser visto como um *workflow* cujo negócio principal é a produção de *software*. Assim, muitos componentes de um ambiente de engenharia de *software* e de um WfMS são semelhantes e podem ser utilizados nos dois contextos. A partir da constatação dessa semelhança o ambiente ExPSEE teve a arquitetura do seu ambiente redefinida de forma compatível com arquitetura da WfMC. Essa arquitetura foi concebida com base em DBC.

Na próxima seção, é apresentada a arquitetura de linha de produto para WfMS proposta por LAZILHA (2002) e é descrito o processo utilizado para seu desenvolvimento.

3.6. A Arquitetura de Linha de Produto para WfMS (LAZILHA, 2002)

A partir da análise da arquitetura genérica e do modelo de referência para WfMS da WfMC (WFMC, 1995), LAZILHA (2002) propôs uma arquitetura para WfMS segundo a abordagem de linha de produto de *software*. O trabalho desenvolvido constituiu o contexto para o componente *WorkflowExecutionMgr*.

O processo de desenvolvimento da arquitetura proposta por LAZILHA (2002) evolve: a exploração do domínio por meio do modelo de referência e da arquitetura genérica para WfMS da WfMC, o método de DBC Catalysis e a linguagem Rapide (CSL, 1997) para simulação e avaliação da arquitetura. O método Catalysis foi utilizado porque é uma abordagem geral para DBC, baseada na UML que envolve conceitos de arquitetura de *software*, *frameworks* e padrões. O objetivo é aplicar um

método de propósito geral, examinando alterações necessárias ao invés de se criar um método específico para linha de produto.

A Figura 3.5 mostra a arquitetura de linha de produto para WfMS proposta por LAZILHA (2002) e em seguida são descritas as principais funcionalidades de cada um dos componentes.

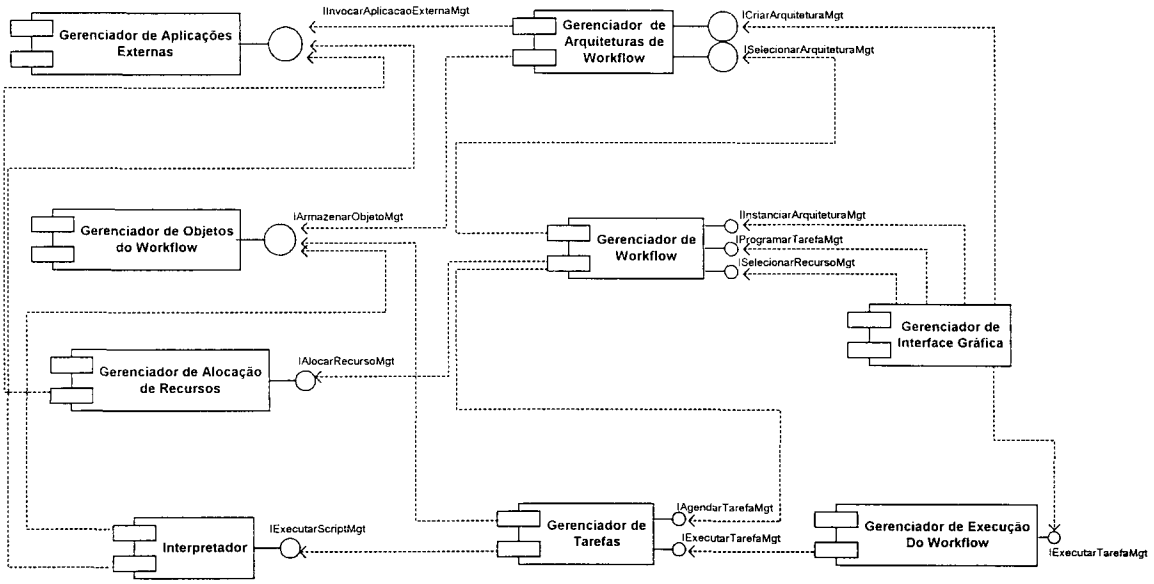

Figura 3.5.: Arquitetura de Componentes Proposta por LAZILHA (2002).

Gerenciador de Workflow: este componente é responsável pela criação e gerenciamento de projetos que incorporam *workflow*s. Os projetos incluem a instanciação e execução de arquiteturas de *workflow*. A cada projeto existe um *workflow* associado. Para cada tipo de objeto existente na arquitetura, instancia-se um objeto no *workflow*. Em seguida, as tarefas do projeto são executadas e gerenciadas, fazendo-se a alocação de recursos e demais tomadas de decisão. Não foram definidos pontos de variação associados a este componente.

Gerenciador de Execução de Workflow: realiza o controle e gerenciamento das tarefas a serem realizadas no *workflow*. Sua principal característica é apoiar a interação com os usuários do WfMS. É através dele que os usuários identificam as suas tarefas no *workflow*.

Gerenciador de Interface Gráfica: é responsável pela interação com o usuário do WfMS. Este componente tem como ponto de variação a interface com o usuário que pode ser via interface gráfica convencional ou via *browser*.

Gerenciador de Arquitetura de Workflow: realiza o controle e gerenciamento da construção e manutenção de arquiteturas de *workflow*. A implementação deste componente suporta as funções relacionadas à definição das arquiteturas de *workflow* e os tipos de objetos relacionados a esta. A utilização deste componente torna a definição de *workflow*s mais flexível. Os pontos de variação associados a este componente indicam o tipo de recurso que pode ser utilizado. Este tipo de recurso pode ser especializado em ator, material e ferramenta e o tipo de ferramenta pode ser especializado em interna ou externa. Diferentes linguagens de programação de processos podem ser apoiadas para a definição das tarefas, que posteriormente serão executadas pelo interpretador.

Gerenciador de Workflow: cria e gerencia os projetos que incorporam *workflows*. Os projetos incluem a instanciação e execução de arquiteturas de *workflow*. Associado a cada projeto existe um *workflow*. Instancia-se um objeto no *workflow* para cada tipo de objeto existente na arquitetura. Em seguida, as tarefas do projeto são executadas e gerenciadas, fazendo-se a alocação de recursos e demais tomadas de decisão. Não existem pontos de variação associados a este componente.

Gerenciador de Tarefas: Este componente é responsável pelo controle e gerenciamento das tarefas e ações a serem realizadas no *workflow*. É através deste componente que os usuários interagem com o sistema gerenciador de *workflow*. O Gerenciador de Tarefas permite que os usuários identifiquem suas tarefas geradas pelo Gerenciador de *Workflow*. Os pontos de variação deste componente são: os tipos de recursos que podem ser utilizados são material, ferramenta e ator; a ferramenta utilizada pode ser do tipo interna ou externa; as tarefas podem ter prioridades de execução diferentes e, os algoritmos utilizados para escalonamento das tarefas também podem variar.

Gerenciador de Alocação de Recursos: componente destinado à alocação dos recursos necessários (ex. alocação de atores, ferramentas, artefatos e definição das datas de execução das tarefas) na realização dos projetos. Além da variabilidade associada ao tipo de recurso e ao tipo de ferramenta, as políticas de alocação de recurso podem variar.

Gerenciador de Aplicações Externas: promove o gerenciamento das aplicações externas, invocadas durante a definição do processo e execução das tarefas. Pontos

de variabilidade incluem as diferentes formas de adaptação da aplicação externa ao sistema gerenciador de *workflows*.

Gerenciador de Objetos do Workflow: é responsável pelo relacionamento com os mecanismos de armazenamento de objetos manipulados pelo sistema. Suas funções trabalham com objetos, que podem ser considerados desde dados de controle do processo, dados relevantes do processo ou, até mesmo, instâncias de *workflow* e tarefas. Todos os componentes pertencentes à arquitetura de componentes proposta utilizam seus serviços. Sua presença torna o restante dos componentes independentes de uma implementação particular de um sistema gerenciador de objetos, garantindo flexibilidade e portabilidade para toda a coleção de componentes existentes. Pontos de variação deste componente incluem os adaptadores para os gerenciadores de bancos de dados.

Interpretador: Interpretas as tarefas definidas para que possam ser executadas. As tarefas que compõem o *workflow* são então programadas utilizando uma linguagem de programação de processos e recursos que permitem a execução cooperativa dessas tarefas. Para a execução dessas tarefas, o gerenciador de tarefas realiza chamadas ao interpretador da linguagem para que este possa executá-las.

Todos os detalhes referentes à arquitetura de linha de produto para WfMS estão disponíveis em (LAZILHA, 2002).

3.7. Considerações Finais

Neste capítulo foram estabelecidos os conceitos básicos da tecnologia de *workflow*. Especialmente sobre os WfMS. Foram também estabelecidos os conceitos e a terminologia necessária para uma melhor fundamentação e compreensão dessa tecnologia. Foi apresentado, também, o projeto ExPSEE que oferece uma plataforma experimental para investigar novas tecnologias sobre processos de *software* e mecanismos para sua modelagem. Os conceitos relacionados ao ExPSEE são importantes para este trabalho devido à correlação existente entre o ambiente de Engenharia de *Software* e WfMS.

Finalmente, foi apresentada a arquitetura de linha de produto para WfMS, proposta por LAZILHA (2002). Essa arquitetura, da qual o componente

WorkflowExecutionMgr faz parte, e o processo utilizado em sua concepção é fundamental na elaboração deste trabalho.

As aplicações dos conceitos apresentadas neste capítulo serão utilizadas no Capítulo 4.

CAPÍTULO 4 – O COMPONENTE GERENCIADOR DE EXECUÇÃO DE WORKFLOW

Um dos objetivos deste trabalho é incrementar o Núcleo de Artefatos para a arquitetura de linha de produto de *software* para WfMS proposta por LAZILHA (2002). O componente *WorkflowExecutionMgr* é um dos artefatos principais do WfMS proposto.

O processo seguido para projetar o *WorkflowExecutionMgr* é equivalente àquele proposto por LAZILHA (2002). Este processo baseia-se na utilização de um método de propósito geral com ênfase em DBC, o Catalysis, com modificações que propiciem a representação de variabilidades, e um mecanismo destinado à validação de arquiteturas.

Para avaliação do componente *WorkflowExecutionMgr* foi implementado um protótipo que aciona os componentes relacionados a este através das interfaces externas promovendo a execução de um *workflow* previamente instanciado.

Os mecanismos utilizados para implementação do protótipo foram escolhidos com base nos estudos de Oliveira (OLIVEIRA JUNIOR, 2002) que pesquisou mecanismos alternativos para se construir uma variante *open source* do ExPSEE (GIMENES, 2001).

Um dos resultados do projeto do *WorkflowExecutionMgr* foi a revisão da arquitetura proposta por LAZILHA (2002), assim este Capítulo apresenta, na Seção 4.1, a revisão desta arquitetura. Na Seção 4.2, está o projeto do componente *WorkflowExecutionMgr*, constituído de uma descrição do componente *WorkflowExecutionMgr* e sua especificação. Na Seção 4.3, é apresentada a avaliação do componente.

4.1. Revisão da Arquitetura para Linha de Produto para WfMS.

Durante o projeto do componente *WorkflowExecutionMgr* foi realizada uma revisão na arquitetura de linha de produto para WfMS proposta por LAZILHA (2002). Assim, foram encontrados alguns problemas. A Figura 4. 1 apresenta a arquitetura com as alterações promovidas.

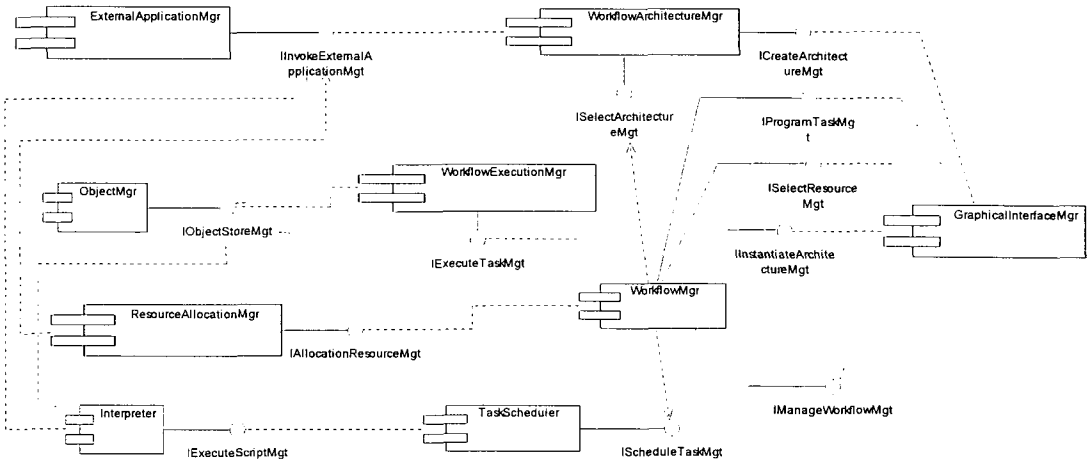

Figura 4. 1.: Arquitetura de Linha de Produto para WfMS - Revisada

Durante a revisão da arquitetura proposta por LAZILHA (2002) (ver Seção 3.6), percebeu-se que o componente *WorkflowExecutionMgr* era parte do componente *WorkflowMgr.* Assim, foram modificados os acessos às interfaces *IExecuteTaskMgt*. Foi removida a dependência que havia entre o componente *WorkflowExecutionMgr* e o componente *GraphicalUserInterface* que só acessará as funcionalidades do componente *WorkflowExecutionMgr* através do componente *WorkflowMgr*. A interface *IExecuteTaskMgt*, passou a ser somente do componente *WorkflowExecutionMgr*, e não mais uma interface compartilhada com o componente *TaskScheduler*. Essas alterações possibilitaram uma separação mais clara das funcionalidades de cada um dos componentes. Deste modo, o *WorkflowMgr* fica encarregado de instanciar a arquitetura (*WorkflowArchitectureMgr*), permitir a alocação de recursos (*ResourceAllocationMgr*), agendamento de tarefas (*TaskScheduler*) e instanciar o *workflow.* O componente *WorkflowExecutionMgr* é responsável pelo gerenciamento da execução a partir do momento em que o *workflow* foi instanciado.

4.2. Projeto do Componente Gerenciador de Execução de Workflow (WorkflowExecutionMgr)

O *WorkflowExecutionMgr* destina-se a executar e gerenciar as tarefas de um *workflow* previamente instanciado. Este gerenciamento se dá através da solicitação de usuários do *workflow* para que determinada tarefa seja colocada em execução ou seja concluída. Quando a execução de uma tarefa é solicitada, o componente verifica se as pré-condições necessárias para a sua execução foram satisfeitas, muda o estado da tarefa, instancia os gerenciadores de tempo, de pré-condições e de transições. No caso de conclusão de tarefa, são verificadas as pós-condições e, se todas foram satisfeitas, muda-se do estado da tarefa. Temos dois tipos de tarefas, as automáticas (não dependem de intervenção humana) e as manuais (que dependem de intervenção humana). Em ambos os casos os procedimentos que serão realizados para permitir ou não as mudanças de estados das tarefas são os mesmos. A diferença está na maneira como é feita a invocação a uma tarefa. A automática é mostrada para o usuário, mas ficam bloqueadas as interações a ela. Na tarefa manual, as interações são permitidas. O bloqueio significa que é o gerenciador quem ativa a execução de uma tarefa e aguarda uma pós-condição que permita que a tarefa seja concluída, sem interferência do usuário do *workflow*.

O projeto do componente *WorkflowExecutionMgr* foi realizado seguindo os seguintes estágios: especificação de requisitos, especificação do sistema, projeto da arquitetura e projeto interno dos componentes. A seguir são apresentados esses estágios.

Por se tratar de um componente que integrará uma arquitetura de linha de produto houve a preocupação em mapear e representar as variabilidades em todos os estágios.

4.2.1. Estrutura para o Processo de Desenvolvimento

Como ferramenta de apoio ao projeto do componente *WorkflowExecutionMgr*, foi utilizado o Rational Rose. Esta ferramenta baseia-se no Processo Unificado da Rational Software (RUP - *Rational Unified Process*) (KRUCHTEN, 2000). Porém, o método Catalysis não possui todos os seus estágios compatíveis com o RUP. Dessa

forma, foi definido um *framework*[1] de projeto para o Catalysis na Rational Rose. Os quatro estágios do Catalysis foram distribuídos entre os três pacotes inerentes a Rational Rose, definindo a estrutura do *framework* da seguinte maneira: *use case view*: especificação de requisitos; *logical view:* especificação do sistema e projeto da arquitetura. Esse é composto das camadas de interface gráfica para o usuário, aplicação e sistema; e *component view:* projeto interno dos componentes.

A Figura 4. 2 mostra o *framework* para o método Catalysis criado na ferramenta Rational Rose e utilizado no projeto do *WorkflowExecutionMgr*.

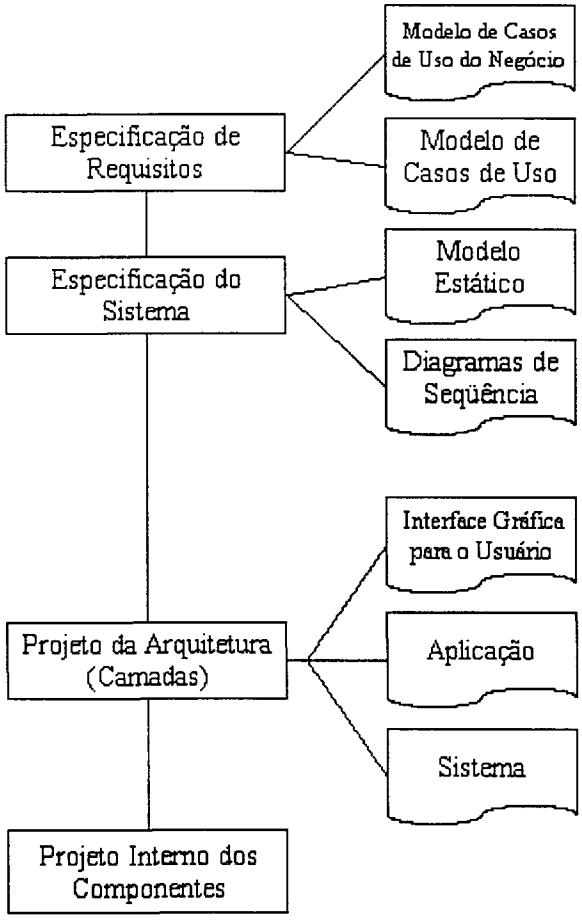

Figura 4. 2.: *Framework* para Catalysis Criado na Ferramenta Rational Rose.

Os retângulos representam as etapas do método Catalysis e as representações de documentos são os artefatos desenvolvidos em cada etapa.

[1] Neste contexto, o *framework* é a estrutura criada na ferramenta para representar o conjunto de visões do processo de desenvolvimento.

Para a representação das variações com a ferramenta Rational Rose, foram criados e adicionados à ferramenta os estereótipos para a representação de casos de uso que apresentam variação, e de atores que apresentam variação. Esses estereótipos são mostrados na Figura 4. 3.

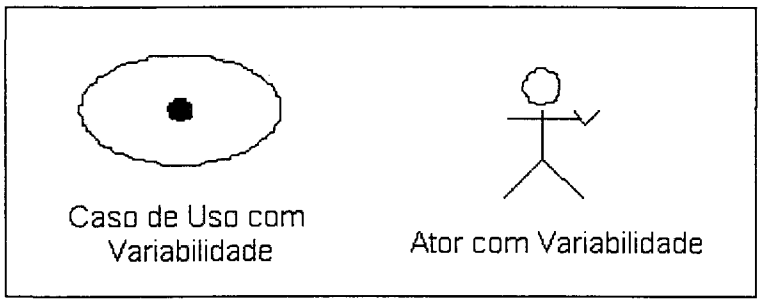

Figura 4. 3.: Estereótipos Adicionados à Ferramenta Rational Rose

O procedimento para a criação e adição destes estereótipos está presente na ajuda *on-line* da própria ferramenta.

A arquitetura envolve um conjunto de componentes. Assim, no projeto de um componente específico, os demais são vistos como atores externos. Quando a variação em um implica em variação no outro o ator externo é marcado como um "V", como acima.

4.2.2. Especificação de Requisitos

A especificação de requisitos foi representada por diagramas de caso de uso da UML. Essa especificação foi dividida em dois modelos: o modelo de negócios e o modelo de casos de uso, em que o segundo é um refinamento do primeiro.

Para a representação das variações este trabalho utiliza a mesma notação utilizada por (LAZILHA, 2002) que estão baseadas nas notações propostas por JACOBSON, GRISS e JONSSON (1997) e MORISIO e TRAVASSOS (2000). A aplicação desta representação das variabilidades pode ser observada na Figura 4. 5.

Nos casos de uso *WorkflowMgr* e *WorkflowExecutionMgr*, a variação segue a proposta de BACHMAN e BASS (2001), que estabelece que quando a variação acontecer na relação entre dois artefatos, ela deve ser representada em ambos.

No relacionamento entre o ator *TaskScheduler* e o caso de uso *WorkflowMgr* indicou-se a variação neste como resultado de variação em *WorkflowMgr*. Esta

variação se dá nos dados, pois o formato das informações enviadas de um componente para outro variará de acordo com a variabilidade instanciada. Por exemplo, se para um determinado produto for instanciada a variabilidade para que o escalonamento das tarefas seja realizado com controle de prioridades, o *TaskScheduler* deverá respeitar a ordem das prioridades na exibição das tarefas para o usuário. Já no caso de um algoritmo de escalonamento sem prioridades, o foco está em exibir as próximas tarefas disponíveis para execução, independente da ordem.

A Figura 4. 4 mostra o diagrama de casos de uso que representa o modelo do negócio em que está inserido o componente *WorkflowExecutionMgr*.

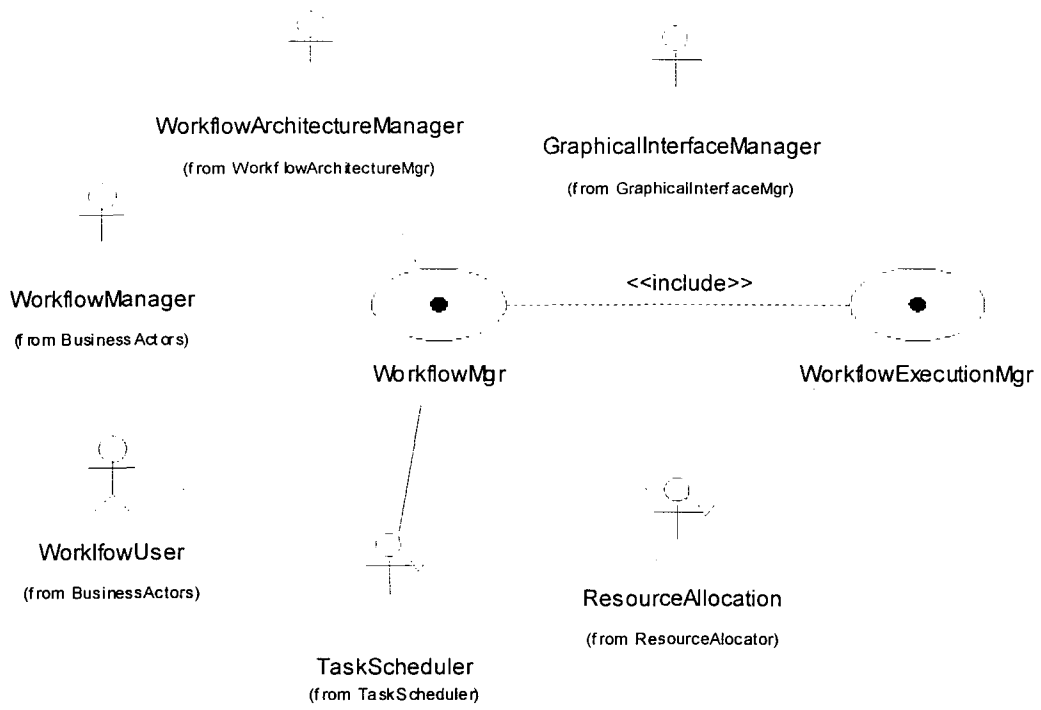

Figura 4. 4.: Modelo de Negócio do componente *WorkflowExecutionMgr*.

Na especificação do negócio, o caso de uso *WorkflowMgr* está associado aos atores *WorkflowArchitectureManager*, *GraphicalInterfaceManager*, *ResourceAllocation* que neste contexto, representam componentes da arquitetura de linha de produto para WfMS (Figura 3.5). É preciso destacar que os demais componentes foram representados como atores (sistemas externos), conforme

proposto pela notação UML. Esta representação é importante, pois mostra que o componente *WorkflowExecutionMgr* é uma parte do caso de uso *WorkflowMgr.* O modelo de casos de uso apresenta uma visão ampla, com o objetivo de apresentar não somente o caso de uso *WorkflowExecutionMgr*, mas também o contexto no qual ele está inserido. Por exemplo, podemos considerar que o caso de uso *WorkflowExecutionMgr* está para o caso de uso *WorkflowMgr* assim como o motor de um automóvel está para o restante do veículo. E neste trabalho, estamos desenvolvendo apenas o motor.

A Figura 4. 5 apresenta o modelo de casos de uso para o componente *WorkflowExecutionMgr*, permitindo visualizar as principais ações e os pontos de variação identificados.

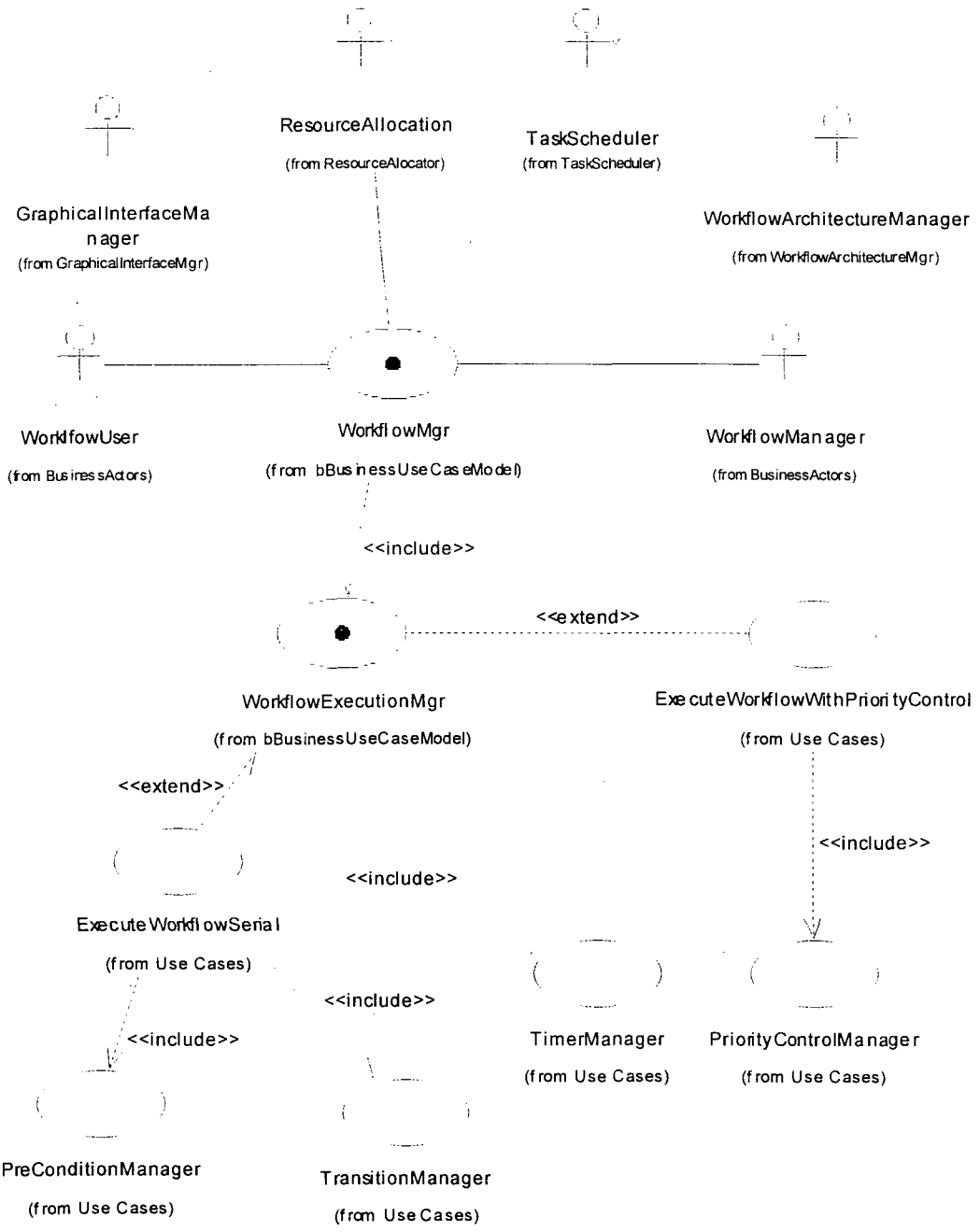

Figura 4. 5.: Modelo de Casos de Uso para o componente WorkflowExecutionMgr.

4.2.3. Especificação do Sistema

Neste estágio do desenvolvimento tratou-se da modelagem da solução de *software* identificada nos modelos obtidos na fase de especificação de requisitos.

A Figura 4. 6 mostra o diagrama de tipos para a arquitetura de linha de produto para WfMS, que constitui o contexto no qual o componente *WorkflowExecutionMgr* está inserido.

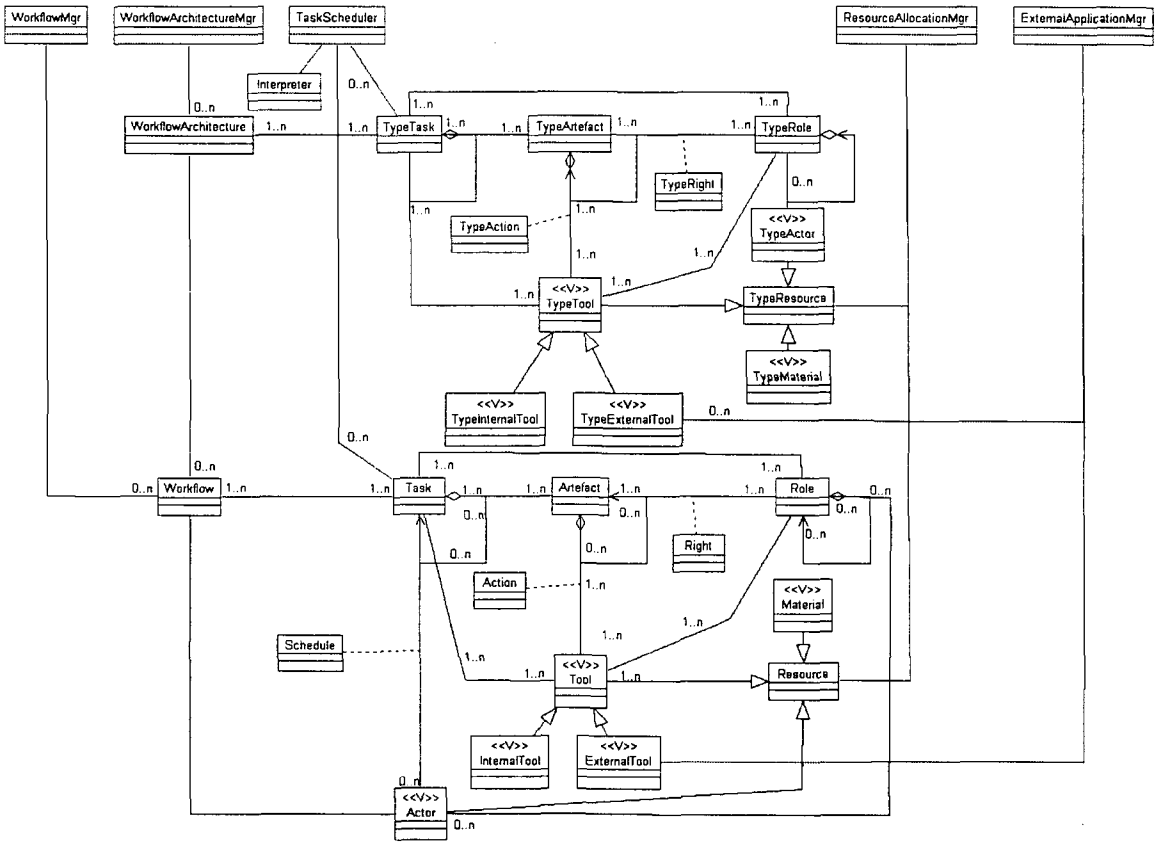

Figura 4. 6.: Diagrama Estático de Tipos para WfMS (LAZILHA, 2002).

Para a representação das variabilidades nos tipos, foi utilizado o estereótipo <<V>>, junto ao nome do tipo conforme proposto por MORISIO e TRAVASSOS (2000) *apud* LAZILHA (2002).

A análise das ações, representada nos diagramas de caso de uso, torna possível a identificação dos tipos, o que possibilita a associação das ações aos tipos relacionados. Devido ao detalhamento das funcionalidades, foram adicionados tipos, como solução de *software*, para dar suporte às funcionalidades do componente *WorkflowExecutionMgr*. Deste modo, foi criado um diagrama de tipos para esse componente (Figura 4. 7), contendo tipos que não estavam presentes no diagrama estático de tipos para WfMS (Figura 4. 6). Nesse diagrama, o tipo *WorkflowUser*, é uma especialização do tipo *Actor*, presente no diagrama estático de tipos para os WfMS.

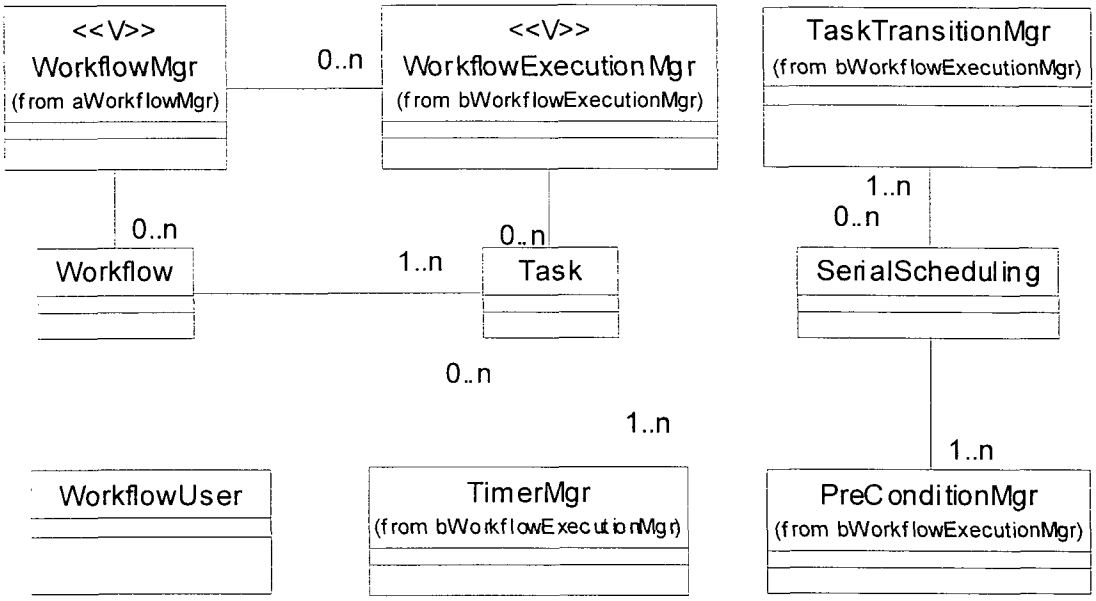

Figura 4. 7.: Diagrama de Tipos para o Componente *WorkflowExecutionMgr.*

A partir do modelo de casos de uso e tipos foi elaborado um diagrama de seqüência para o caso de uso *WorkflowExecutionMgr.* A partir deste ponto foi considerada a solução de um algoritmo de escalonamento serial. Este foi o algoritmo escolhido para a demonstração das funcionalidades do componente *WorkflowExecutionMgr.* Este diagrama é mostrado na Figura 4. 8. É importante ressaltar que neste diagrama o *WorkflowMgr* e o *WorkflowExecutionMgr* são componentes, e não objetos como tradicionalmente é representado nos diagramas de seqüência.

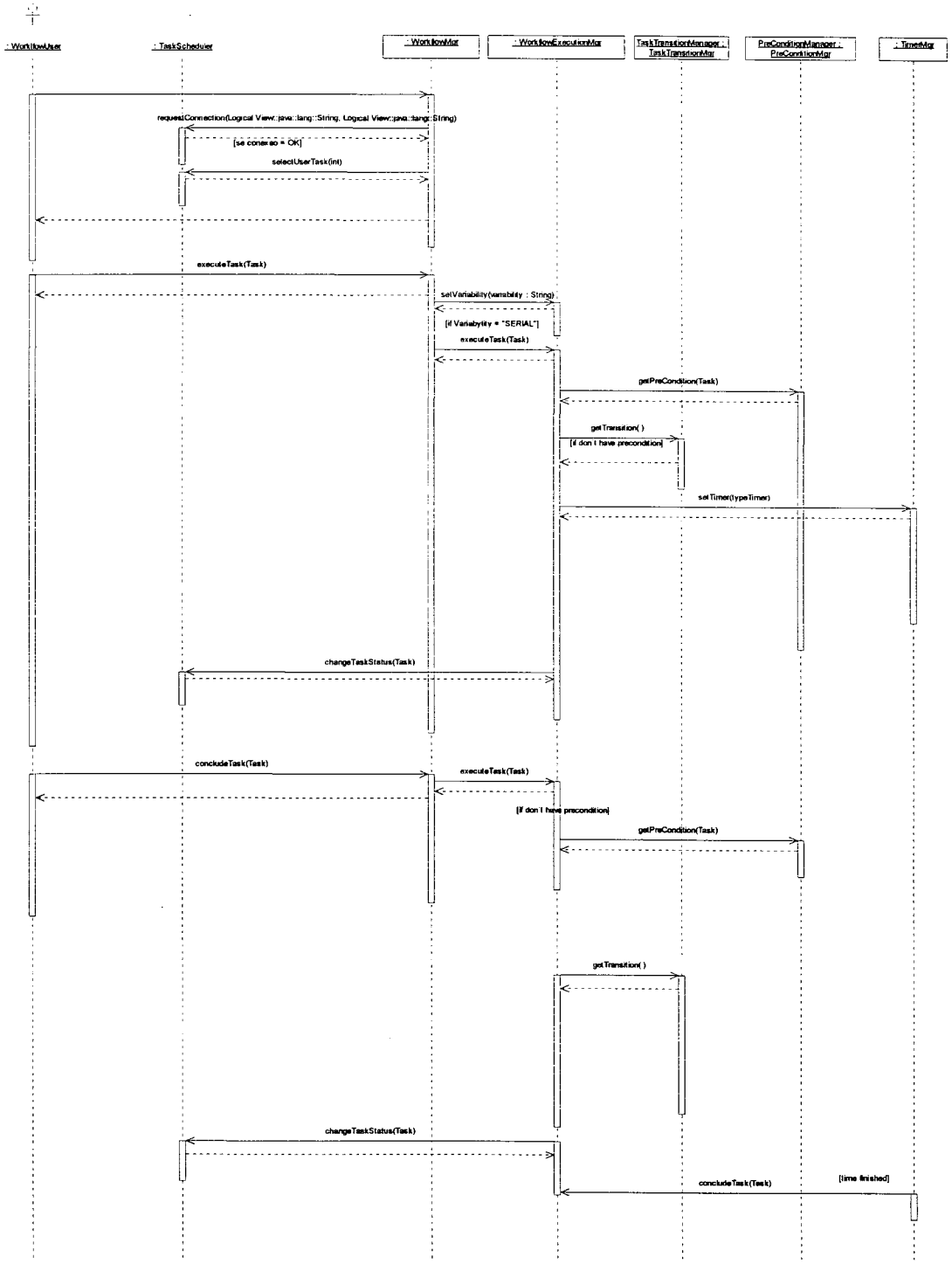

Figura 4. 8.: Diagrama de Seqüência para Escalonamento Seqüencial.

As principais ações identificadas no diagrama de seqüência estão relacionadas às tarefas e seus estados. Assim, a Figura 4. 9 apresenta o diagrama de estados das tarefas, que é o mesmo utilizado no *process-manager* do ambiente ExPSEE (GIMENES, 2001).

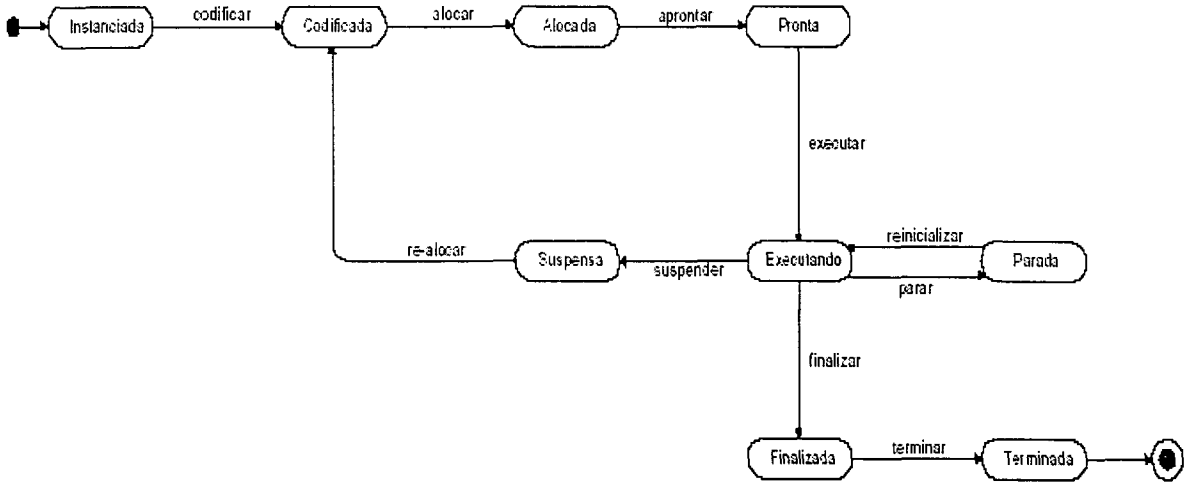

Figura 4. 9.: Diagrama de Estados das Tarefas (GIMENES, 2001).

Após a instanciação, o *WorkflowExecutionMgr* deve guiar a execução das tarefas através da seqüência de estados definida no diagrama.

O diagrama representa todos os estados, pelos quais cada tarefa deve ou pode passar durante sua execução, desde o estado inicial – instanciada – até o estado final – terminada. Para que as transições possam ocorrer, um conjunto de pré-condições, próprio para cada uma delas, deve ser satisfeito. A seguir encontra-se uma descrição do diagrama de estados da Figura 4. 9, bem como do conjunto de pré-condições necessários para se alcançar cada um dos estados.

Cada um dos estados apresentados é descrito a seguir:

- ***Estado instanciada:*** a tarefa neste estado tem, no mínimo, suas características especificadas através do seu: tipo, nome e descrição. Além disso, é necessário que ela tenha sua estrutura estabelecida através de seus relacionamentos com outros objetos do processo: artefatos necessários e a serem produzidos, ferramentas a serem utilizadas e cargos requeridos. Estes relacionamentos devem estar de acordo com os definidos na arquitetura de

processo. A tarefa é dita instanciada quando têm suas características e estrutura estabelecida. Porém, ela ainda não está pronta (disponível) para ser executada.

- ***Estado codificada:*** a tarefa neste estado deve satisfazer as condições do estado anterior e o ter seu procedimento instanciado. Neste caso a tarefa é dita codificada com o seu procedimento pronto para ser executado. Porém, ela ainda não tem todos os recursos necessários para ser executada, e portanto, permanecerá neste estado até que tenha estes recursos alocados.
- ***Estado alocada :*** além das condições do estado anterior, todos os recursos necessários para sua execução alocados devem estar alocados. Assim, é necessária a definição de seu tempo de execução (uma data de início e término reais para a sua realização), a alocação de um ou mais usuários do *workflow*, Neste caso a tarefa é dita alocada e dispõe dos recursos necessários à sua realização alocados para ela. Porém, ela ainda pode não ter todas as pré-condições necessárias para sua execução, pois poderá não ter todos os artefatos necessários nos estados prontos ou alguma tarefa precedente não terminada. Neste estado a tarefa é integrada à agenda dos usuários do workflow que devem executá-la.
- ***Estado pronta:*** a tarefa neste estado deve conter, além das condições do estado anterior, os artefatos necessários disponíveis e em estado satisfatório, e se existirem, suas tarefas precedentes prontas, e suas pré-condições satisfeitas. Neste caso, a tarefa é dita pronta e detém todas as condições necessárias e suficientes para sua execução. Sua permanência neste estado depende dos usuários que devem executá-la a partir de suas agendas de tarefas.
- ***Estado executando:*** quando um ator escolhe uma tarefa em sua agenda de tarefas para execução, esta passa ao então ao estado executando.
- ***Estado parada:*** uma tarefa pode entrar no estado parada a qualquer momento, pela ação voluntária do gerente de *workflow*, no entanto, deve-se guardar o contexto no qual esta tarefa foi parada para que ela possa ser reinicializada corretamente e voltar para o estado executando.

- ***Estado suspensa:*** uma tarefa pode passar ao estado de suspensa somente pela ação do gerente de *workflow*, caso este deseje ou necessite alocar as ferramentas e/ou atores desta tarefa a outras tarefas. Quando uma tarefa é suspensa, as ferramentas e/ou atores alocados a ela devem ser liberados. Neste caso a tarefa volta para o estado codificado.
- ***Estado finalizada:*** uma tarefa entra no estado finalizada após ser executada. Para que uma tarefa possa ser finalizada é necessário que suas pós-condições estejam satisfeitas. Neste estado, a tarefa fica esperando até que suas sub-tarefas (hierarquicamente dependentes), se existirem, possam ser terminadas.
- ***Estado terminada:*** uma tarefa entra no estado terminada depois que todas suas sub-tarefas, se existirem, estiverem terminadas. Caso a tarefa não possua sub-tarefas, então ela passa automaticamente do estado finalizada para o estado terminada.

Pode ser necessária a modificação de uma tarefa, tendo que fazê-la voltar a um estado anterior ao seu. No entanto, este é um problema complexo, e pode ser tema de um novo trabalho a ser desenvolvido. Para este trabalho as transições de estados mais relevantes são a partir do estado "pronta" até o estado "finalizada". Pois o componente *WorkflowExecutionMgr* tem por objetivo a execução das tarefas. Todos os processos anteriores à execução são de responsabilidade de outros artefatos da arquitetura para WfMS.

4.2.4. Projeto da Arquitetura

O projeto da arquitetura de um sistema de *software* visa definir suas estruturas gerais descrevendo os elementos que compõem os sistemas e as interações entre estes apoiando também questões importantes de projeto como, por exemplo, a organização do sistema como uma composição de componentes. Conforme o processo proposto por LAZILHA (2002), o *framework* para o Catalysis utilizado neste trabalho inclui o projeto da arquitetura que está dividido nas camadas de interface com o usuário, camada de aplicação e camada de sistema.

Para o componente *WorkflowExecutionMgr*, devido a seu tamanho, não foi necessária a definição de uma arquitetura de componentes. Deste modo, passou-se diretamente para a etapa do projeto interno do componente, apresentado na Seção 4.2.5.

4.2.5. Projeto Interno do Componente

Neste estágio os componentes individuais são refinados até o nível de interfaces, classes ou componentes pré-existentes em uma linguagem de programação. Para que isto ocorra, a implementação deve satisfazer os requisitos funcionais e não funcionais definidos na especificação do sistema, assim como seguir a arquitetura definida.

Para o componente *WorkflowExecutionMgr*, foi elaborado um diagrama de classes. Esse diagrama é mostrado na Figura 4.10.

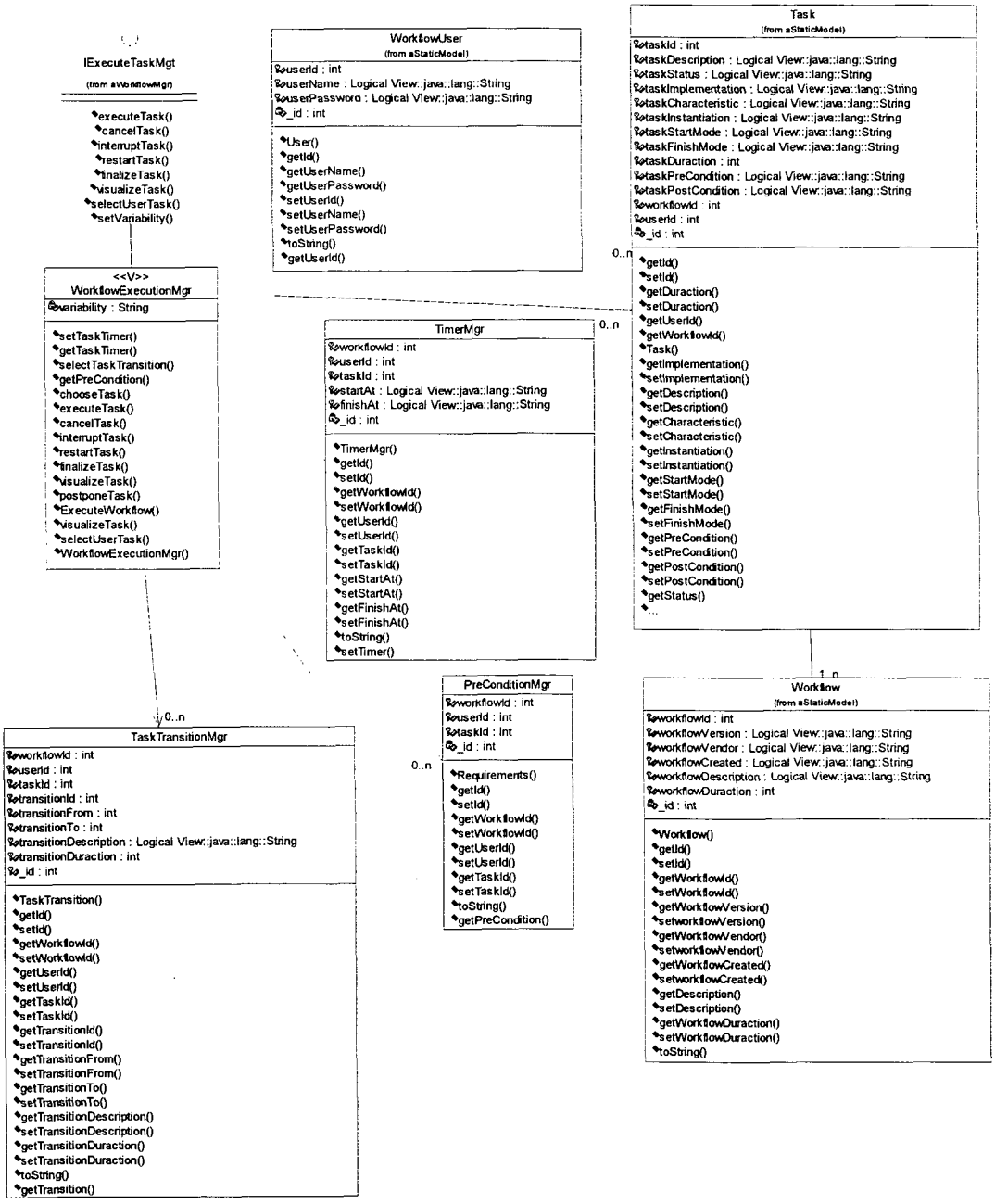

Figura 4. 10.: Projeto Interno do Componente *WorkflowExecutionMgr*.

Para o projeto interno do componente *WorkflowExecutionMgr* foram criados atributos e métodos para cada classe, detalhando-os de forma a possibilitar que o código fonte para o protótipo fosse gerado a partir deste diagrama utilizando-se a ferramenta Rational Rose.

A Figura 4.11, apresenta uma representação do componente *WorkflowExecutionMgr*. São apresentadas deste modo, as classes que constituem este componente e, em destaque, a classe *IExecuteTaskMgt* que é a interface provida pelo componente. Para esta representação foram omitidos os atributos e os métodos presentes no diagrama de classes, mostrado na Figura 4. 10.

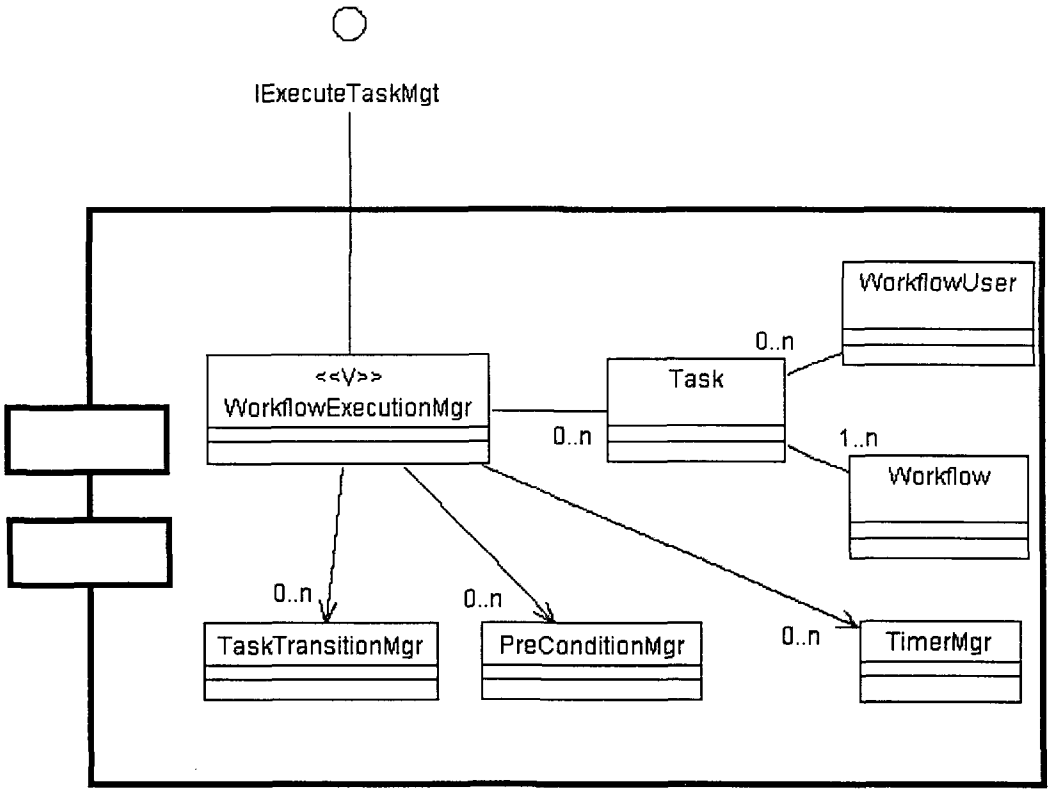

Figura 4. 11.: O componente *WorkflowExecutionMgr*.

4.2.6. Especificações em OCL

Para formalizar o funcionamento das interfaces e métodos do componente *WorkflowExecutionMgr* foi utilizada a OCL (WARMER; KLEPPE, 1998) que é uma linguagem para especificar restrições e outras expressões associadas aos modelos da UML. A OCL oferece o formalismo necessário para a especificação de sentenças bem definidas nos primeiros estágios do processo de desenvolvimento.

Para as interfaces do componente foram descritas restrições através de pré e pós-condições.

4.2.6.1. Interface Externa: IExecuteTaskMgt

executeTask:
pre : -- task must be in state 4 (Ready)
self.task.status = 4
post : -- task must be in state 6 (Executing)
self.task.status = 6

cancelTask
pre : -- task must be in state 6 (Executing)
self.task.status = 6
post : -- task must be in state 9 (Terminated)
self.task.status = 9

interruptTask
pre : -- task must be in state 6 (Executing)
self.task.status = 6
post : -- task must be in state 5 (Suspended)
self.task.status = 5

restartTask
pre : -- task must be in state 5 (Suspended)
self.task.status = 5
post : -- task must be in state 6 (Executing)
self.task.status = 6

finalizeTask
pre : -- the status must be in state finalized
self.task.status = 8
post : -- the status must be in state terminated
self.task.status = 9

visualizeTask
pre: -- The role name must be "*WORKFLOW* MANAGER" or "*WORKFLOW* USER".
self.role.roleName = "*WORKFLOW* MANAGER" or self.role.roleName =
"SOFTWARE ENGINEER"
post:
if self.role.roleName = "WORKFLOW MANAGER" then
result = Task.allInstances
else
if self.role.roleName = "WORKFLOW USER" then
result = Task.allInstances->Role.includes("WORKFLOW USER")
endif
endif

selectUserTask
pre: -- The role name must be "WORKFLOW MANAGER" or "WORKFLOW USER".

```
 self.role.roleName = "WORKFLOW MANAGER" or self.role.roleName =
"SOFTWARE ENGINEER"
post:
 if self.role.roleName = "WORKFLOW MANAGER" then
 result = Task.allInstances
 else
 if self.role.roleName = "WORKFLOW USER" then
  result = Task.allInstances->Role.includes("WORKFLOW USER")
 endif
 endif
```

setVariability
pre: --none
post: self.variability = WorkflowMgr.variability.value

4.2.6.2. Métodos:

WorkflowExecutionMgr

setTaskTimer
pre: -- Task must Exist
pos: -- self.startAt = startAt
self.finishAt = finishAt

getTaskTimer
pre: -- none
post: -- result = timer

getPreCondition
pre: -- none
post: -- if preCondition.task
then result. = preCondition.task
else result = null

postponeTask
pre: -- Task must be in state 6 (Executing)
post :-- Task must be in state 5 (Suspended)
self.task.status = 6

Existem muitos métodos com objetivo único de atribuir/fornecer valores de atributos, presentes nas classes *TimerMgr*, *PreConditionMgr*, *TaskTransitionMgr* para os quais não foram necessárias descrições de restrições.

4.3. Avaliação do Componente

Para avaliação do componente *WorkflowExecutionMgr* foi implementado um protótipo que aciona os componentes relacionados a este através das interfaces externas promovendo a execução de um workflow previamente instanciado.

Para a implementação do protótipo foram escolhidos os seguintes mecanismos, separados em categorias.

- ***Plataforma de Execução Linux:*** esta plataforma disponibiliza os mecanismos necessários para execução.
- ***Linguagem de programação Java***: esta linguagem é de domínio público, totalmente orientada a objetos, e com um extenso *toolkit*, o *Swing* por exemplo, que permite a implementação de interfaces gráficas com o usuário através de vários componentes dinâmicos. Além disso, Java possui mapeamento para mecanismos de comunicação interprocessos existentes como o padrão CORBA (OMG, 2002) e/ou RMI (SUN, 2002).
- ***Construção de Interface com o Usuário (GUI):*** o componente *WorkflowExecutionMgr* não apresenta uma interface própria para o usuário, seus serviços são apresentados ao usuário através da interface para o componente *TaskScheduler*, deste modo, foi necessária a construção de uma interface gráfica para este componente. Para a construção desta interface foi utilizado o *toolkit Swing* (SUN, 2002) pertencente à plataforma Java 2.
- ***Padrão de comunicação interprocessos CORBA:*** o componente *WorkflowExecutionMgr* está inserido em uma arquitetura distribuída, deste modo, o padrão CORBA (OMG, 2002) é adequado para a comunicação interprocessos, utilizando para isso o JacORB (JACORB, 2002) que é um ORB (*Object Request Broker*) implementado em Java.
- ***Sistema Gerenciador de Banco de Dados:*** o sistema gerenciador de banco de dados escolhido foi o MySQL (MYSQL, 2003). No entanto, para realizar o mapeamento dos objetos para uma base de dados relacional ou objeto-relacional foi necessário utilizar um *framework* que executasse tal tarefa. O *framework* escolhido foi o ObJectBridge (JAKARTA, 2002) que faz parte do projeto Jakarta.

4.3.1. Instanciação das Variabilidades

O principal ponto de variação identificado no componente *WorkflowExecutionMgr* é a possibilidade de executar diferentes algoritmos de escalonamento. Esta variabilidade é uma instância de várias alternativas, descrita na Seção 2.2. A causa desta variabilidade está nos dados. Considerando o *WorkflowExecutionMgr*, pode-se instanciar várias alternativas de algoritmos de escalonamento, se for escolhido um algoritmo de escalonamento com controle de prioridades, haverá a necessidade de enviar os dados de forma diferente daqueles enviados pelo algoritmo de escalonamento serial para os outros componentes.

Para que a variabilidade fosse instanciada, optou-se pela criação de um arquivo XML de configuração que contêm os parâmetros necessários para que a aplicação instancie o algoritmo escolhido. Um exemplo de como é mapeada a variabilidade para o componente *WorkflowExecutionMgr* é apresentado na Figura 4. 12.

```
<!–Variability for WorkflowExecutionMgr –->

  <variabilityDescriptor>

<variability.class>WorkflowExecutionMgr</variability.class>
    <variability.name>algoritmoEscalonamento</field.name>
    <variability.value>SERIAL</variability.value>
  </variabilityDescriptor>
```

Figura 4. 12: Exemplo de Código XML para Mapear a Variabilidade.

Foi escolhida a utilização deste arquivo de configuração por apresentar maior flexibilidade em se instanciar diferentes pontos de variabilidade para diferentes componentes.

Instanciação das variabilidades pode ser facilitada com a criação de um programa para este fim capaz de gerenciar este arquivo XML.

Para promover a instanciação das variabilidades, no momento em que o produto é instanciado executa-se o método setVariability() presente na interface *IExecuteTaskMgt* do componente *WorkflowExecutionMgr* identificando qual algoritmo de escalonamento será utilizado.

A instanciação das variabilidades está mais relacionada com o plano de produção do produto, descrito na Seção 2.3.2, portanto fora do domínio deste trabalho.

4.3.2. Interface com o Usuário

A principal interface com o usuário está diretamente relacionada ao componente *TaskScheduler* e mostra as tarefas que estão disponíveis para o *WorkflowUser*, a Figura 4. 13 mostra esta interface.

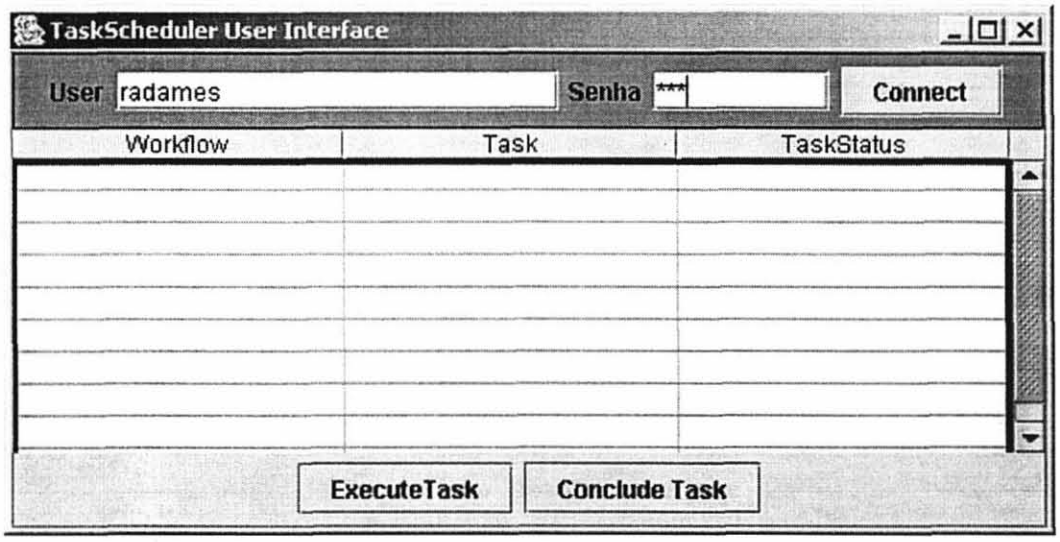

Figura 4. 13.:Interface com o Usuário: TaskScheduler

Ao efetuar o *login*, serão selecionadas todas as tarefas que estão no estado "pronta" ou "executando". O usuário pode, então, escolher uma das tarefas e acionar o botão *ExecuteTask* para iniciar a execução da tarefa ou *ConcludeTask* para informar que determinada tarefa foi concluída. Após a conclusão de uma tarefa, serão listadas as próximas tarefas disponíveis para execução.

4.3.3. Construção da Base de Dados Relacional

A base de dados relacional implementada para o componente, restringiu-se às entidades necessárias para o funcionamento do componente. As estruturas destas entidades são apresentadas no Anexo 2.

Os atributos descritos para as entidades são restritos às necessidades do protótipo.

4.3.4. Mapeamento dos Objetos para a Base de Dados Relacional

Para realização do mapeamento dos objetos para uma base de dados relacional é utilizado um arquivo XML que informa como deve ser realizada a transição dos objetos.

O arquivo XML, descreve inicialmente o mapeamento para o banco de dados relacional onde são informados, o *driver* a ser utilizado, o endereço do local onde o servidor está ativo, o nome do banco de dados, o nome do usuário e a senha.

```
<JdbcConnectionDescriptor id="default">
    <dbms.name>MySQL</dbms.name>
    <driver.name>org.gjt.mm.mysql.Driver</driver.name>
    <url.protocol>jdbc</url.protocol>
    <url.subprotocol>mysql</url.subprotocol>
    <url.dbalias>//200.111.111.203/WfMS</url.dbalias>
    <user.name>rdms</user.name>
    <user.passwd>123</user.passwd>
</JdbcConnectionDescriptor>
```

Após terem sido realizadas as descrições dos elementos necessários para a conexão do banco de dados, são relacionadas todas as entidades às respectivas classes, e cada atributo na entidade do banco de dados deve ter o seu correspondente na classe relacionada a ela. A seguir, está um exemplo do mapeamento da classe *WorkflowUser* para a entidade *User*, representada na Figura 4.14, cuja descrição está na Seção 4.3.3.

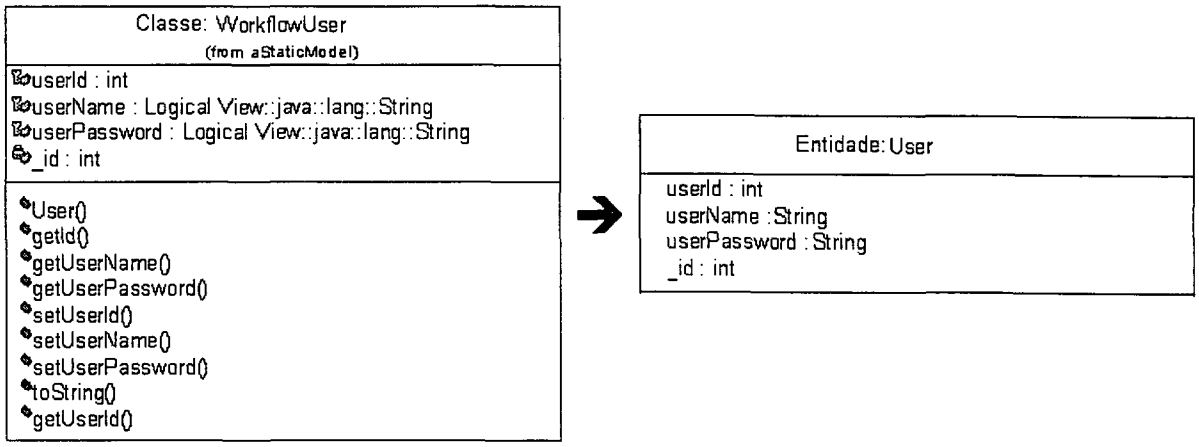

Figura 4.1.4::.Classe *WorkflowUser* mapeada para Entidade *User*.

```
<!-- Definitions for User -->
   <ClassDescriptor id="1">
    <class.name>WorkflowUser</class.name>
    <table.name>User</table.name>

    <FieldDescriptor id="1">
      <field.name>userId</field.name>
      <column.name>userId</column.name>
      <jdbc_type>INTEGER</jdbc_type>
      <PrimaryKey>true</PrimaryKey>
     </FieldDescriptor>

     <FieldDescriptor id="2">
      <field.name>userName</field.name>
      <column.name>userName</column.name>
      <jdbc_type>VARCHAR</jdbc_type>
     </FieldDescriptor>


     <FieldDescriptor id="3">
      <field.name>userPassword</field.name>
      <column.name>userPassword</column.name>
      <jdbc_type>VARCHAR</jdbc_type>
     </FieldDescriptor>
   </ClassDescriptor>
```

As descrições de todas as classes e entidades, utilizadas no protótipo do componente *WorkflowExecutionMgr*, são apresentadas no anexo I.

4.4. Análise dos Resultados de Avaliação

Através da revisão da arquitetura foi possível identificar de forma mais clara as funcionalidades dos componentes *WorkflowExecutionMgr* e *WorkflowMgr*. As alterações promovidas contribuem para o aperfeiçoamento da arquitetura de linha de produto proposta por LAZILHA (2002).

A implementação do protótipo permitiu que fosse visualizada a execução de um *workflow* do ponto de vista do usuário do *workflow*, o que demonstra que as funcionalidades requeridas para o componente foram satisfeitas. O processo de implementação, auxiliou também no refinamento de pormenores não percebidos durante a fase de especificação do componente *WorkflowExecutionMgr*. Deste modo, de forma interativa e incremental, o projeto foi refinado.

Para a validação do componente, como ele está inserido em uma arquitetura, foi necessária a implementação de algumas funcionalidades de outros componentes. Por exemplo, temos a interface gráfica para o *TaskScheduler*, mecanismos de para instanciação do componente presentes no componente *Workflow*Mgr, métodos dos objetos *Task*, *User*, *Workflow*. Estes elementos contribuirão, possivelmente, na realização de outros trabalhos relacionados à arquitetura de linha de produto proposta por LAZILHA (2002).

O componente *WorkflowExecutionMgr* apresentou um número menor de variabilidades em relação ao que era esperado no inicio do projeto. A ausência de outros componentes já implementados dificultou o mapeamento das possíveis variações e como essas variações afetariam outros componentes.

4.5. Considerações Finais

Este Capítulo apresentou a revisão da arquitetura de linha de produto para WfMS proposta por LAZILHA (2002), descreveu e justificou as alterações realizadas. Este processo resultou em uma apresentação mais clara das funcionalidades dos componentes *WorkflowExecutionMgr e WorkflowMgr*.

Foi apresentado o *framework* criado na ferramenta Rational Rose para sua utilização com o método Catalysis.

Mostrou, também, o projeto do componente *WorkflowExecutionMgr*, de acordo com o processo proposto por LAZILHA (2002). Esse processo é composto da

especificação de requisitos (na qual foram produzidos os modelos de casos de uso), da especificação do sistema (com o diagrama de tipos e diagrama de seqüência e diagrama de estados das tarefas), do projeto da arquitetura (no qual, para este projeto, não foi produzido nenhum artefato) e do projeto interno dos componentes (que inclui um diagrama de classes e as especificações em OCL para o componente *WorkflowExecutionMgr*).

Baseado no diagrama de classes, tornou-se possível à geração de código JAVA (fonte) através da ferramenta Rational Rose. Para validar o componente *WorkflowExecutionMgr,* a partir do código fonte, foi implementado um protótipo. Os mecanismos utilizados para a implementação são a linguagem de programação JAVA (SUN, 2002), o gerenciador de banco de dados MySQL (MYSQL, 2003) e o mecanismo para mapear os objetos para uma base de dados relacional *ObJectBridge* (JAKARTA, 2002). Os resultados da implementação do protótipo são a visualização de um *workflow* sendo executado, o que demonstra a realização das funcionalidades do componente e suas interfaces, e o refinamento do projeto do componente *WorkflowExecutionMgr*, de forma interativa e incremental, refinando e acrescentando elementos não percebidos antes da fase de implementação.

5. CAPÍTULO – CONCLUSÕES E TRABALHOS FUTUROS

O trabalho aqui apresentado contribui para a melhoria do processo de desenvolvimento de WfMS com apoio da abordagem de linha de produto.

Esta dissertação apresenta o projeto do componente *WorkflowExecutionMgr*, que faz parte da arquitetura proposta por LAZILHA (2002) para WfMS. A principal funcionalidade deste componente é a execução de um *workflow* previamente instanciado através do gerenciamento e execução de suas tarefas.

O desenvolvimento do componente seguiu o processo proposto por LAZILHA (2002) para o desenvolvimento de arquiteturas de software para linha de produto. Esse processo baseia-se em um método para DBC com a adição de alguns estereótipos para representar variabilidade. A implementação de um protótipo auxiliou no processo de avaliação do componente *WorkflowExecutionMgr*.

Na literatura, encontramos propostas de métodos, que integram os conceitos de DBC e linha de produto de *software*. O método Kobra (ATKINSON; BAYER; MUTHING, 2000), por exemplo, considera o desenvolvimento e a manutenção de um *framework* com as possíveis alternativas de realização da arquitetura de produtos membros. Isso é bastante similar ao processo proposto por LAZILHA (2002). Porém, esse método vem sendo amadurecido praticamente em paralelo com o processo aqui proposto.

Existem várias técnicas para o desenvolvimento de linhas de produto, algumas delas são baseadas em engenharia de domínio, que se mostram, menos eficientes na representação de arquiteturas e componentes. Os métodos de DBC podem ser usados no processo de desenvolvimento de linha de produto para reduzir o intervalo entre a análise do domínio, arquitetura e projeto interno dos componentes.

O método de desenvolvimento utilizado neste trabalho está baseado nos conceitos de DBC e linha de produto. Portanto, oferece-se a possibilidade de utilizar um método mais próximo dos amplamente conhecidos ao invés de um dos métodos de linha de produto relativamente novos.

As principais contribuições deste trabalho são: o projeto do componente *WorkflowExecutionMgr*, que incrementa o núcleo de artefatos para a linha de produto para WfMS, a análise das variabilidades, estabelecendo como o

componente *WorkflowExecutionMgr* pode ser instanciado para os membros da família de produtos, a revisão da arquitetura para a linha de produto para WfMS proposta por LAZILHA (2002) e a avaliação do processo proposto por LAZILHA (2002). A utilização desse processo facilitou o desenvolvimento do componente *WorkflowExecutionMgr*.

O método Catalysis guiou o projeto do componente e as modificações propostas permitiram a definição das variabilidades em todos os estágios do desenvolvimento.

A validação do componente *WorkflowExecutionMgr,* através da implementação de um protótipo, mostrou-se satisfatória, pois demonstrou o comportamento da execução de um *workflow* do ponto de vista do usuário do *workflow*. Isso mostra que as funcionalidades do componente foram satisfeitas. A implementação do protótipo permitiu também o aperfeiçoamento de interfaces e refinamento do projeto com detalhes provenientes da fase de implementação. O componente *WorkflowExecutionMgr* é o primeiro componente a ser desenvolvido para a linha de produto para WfMS proposta por LAZILHA (2002).

A reutilização do componente *WorkflowExecutionMgr* em outros sistemas está vinculada à utilização do componente *WorkflowMgr*, pois o *WorkflowExecutionMgr* é parte deste componente.

Dentre os trabalhos futuros provenientes deste esforço, pode-se destacar o projeto e implementação dos demais componentes da arquitetura, de modo que no futuro possam ser gerados produtos a partir do núcleo de artefatos para a linha de produto para WfMS. O estabelecimento do processo de produção de aplicações da família de produtos. Neste trabalho utilizou-se um arquivo XML para a instanciação das variabilidades. No entanto, é necessária a formalização de um mecanismo mais específico, que incorpore o modelo de decisão.

REFERÊNCIAS

ATKINSON, C.; BAYER, J.; MUTHIG, D. **Component-Based Product Line Development, The Kobra Approach.** Proc. of the 1nd INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, Pittsburgh, 2000.

BACHMANN, F.; BASS, L. **"Managing Variability in Software Architectures"**, Sysmposium on Software Reusability, Toronto, Canada. 18 – 20 may 2001.

BAYER, J; FLEGE, O; KNAUBER, P.; LAGUA, R.; MUTHING, D.; SCHMID, K.; WIDEN, T. **PuLSE: A methodology to develop software product lines.** Proc. of the 1nd SYMPOSIUM ON SOFTWARE REUSABILITY (SSR99), Los Angeles, 1999.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Language Users Guide.** Addison-Wesley Publishing Company, 1999.

BOSCH, J. **Design & Use of Software Architectures: adopting and evolving a product-line approach**. Great Britain: Ed. Addison Wesley, 2000. 354 p.

CLEMENTS, P. **Software Product Line Fundamentals.** Addison- Wesley, 2001.

CLEMENTS, P.; NORTHROP, L. M. **Software Product Lines: Practices and Patterns.** SEI Series in Software Engineering, Ed. Addison Wesley. 563 p, 2001.

CSL - Computer Science Lab, **DRAFT Guide to Rapide 1.0, Language Reference Manuals**, Rapide Design Team – Program Analysis and Verification Group, Stanford University, July 1997. 80p.

D'SOUZA, D., WILLS, A. **Objects, Components and Frameworks with UML – The Catalysis Approach**, Addison-Wesley Publishing Company, 1999. 816 p.

DENGI, E. C. **An Improved Model and Architecture of Workflow Process Management.** The Graduate School of Natural and Applied Sciences of the Middle East Technical University, 1998.

GIMENES, I. M. S.; HUZITA, E. H. M.; BARROCA, L. CARNIELLO, A. **O Processo de Desenvolvimento Baseado em Componentes Através de Exemplos.** ERI 2000 VIII ESCOLA DE INFORMÁTICA DA SBC-SUL, Foz do Iguaçu, Ed. Raul Ceretta, 2000. 252 p.

GIMENES, I. M. S.; HUZITA, E. H. M.; TAKANO, E. T.; STEINMACHER, I. F. **ExPSEE - An Experimental Process-centred Software Engineering Environment, Relatório Final.** PPG-UEM, Maringá-PR, 2001.

GIMENES, I. M. S., TRAVASSOS, G. H. **O Enfoque de Linha de Produto para Desenvolvimento de Software**. Florianópolis, SC: SBC, 2002. 34 p. Trabalho apresentado na Jornada de Atualização em Informática, 2002.

GRISS, M. **Integrating Feature Modeling with RSEB.** Proc. of the 5nd International Conference on Software Reuse (ICSR-5), ACM/IEEE, Victoria, Canadá, Junho de 1998.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. **Rational Unified Software Development Process**. [S.l]: Addison Wesley, 1999.

JACOBSON, I.; GRISS, M.; JONSSON, P. **Software Reuse – Architecture Process and Organization for Business Success**. New York: Ed. Addison-Wesley, 1997. 497 p.

JACORB. **The free Java Implementation of OMG's CORBA Standard.** Disponível em: <http://www.jacorb.org>. Acessado em Jul. 2002.

JAKARTA PROJECT. **ObJectRelationBridge**. Disponível em: <http://jakarta.apache.org/ojb>. Acesso em 20 jul. 2002.

KANG, K. **Feature-Oriented Domain Analysis (FODA) Feasibility Study** *(CMU/SEI-90-TR-21, ADA 235785)*. Pittsburgh, PA: SEI CMU, 1990.

KRUCHTEN, P. **RUP - The Rational Unified Process**. 2. ed. EUA: Addison-Wesley Publishing Company, 2000.

LAZILHA, F. R. **Uma Proposta de arquitetura de Linha de Produto para Workflow Management Systems**. Dissertação. Universidade Federal do Rio Grande do Sul. Instituto de Informática. Porto Alegre. Janeiro de 2002.

MORISIO, M., TRAVASSOS, G.H., Stark, M. **Extending UML to Support Domain Analysis**, IEEE International Conference on Automated Software Engineering – ASE'00. Grenoble, France, 2000.

MYSQL. **MySQL**, Disponível em: www.mysql.com. Acesso em 20 jun. 2003.

OLIVEIRA JUNIOR, E. A. GIMENES, I.M.S. **Portando o ExPSEE para a Plataforma Linux, Relatório Semestral de Iniciação Científica**, PPG-UEM, Maringá, 2002.

OMG - Object Management Group, Disponível em: <http://www.omg.org>, 2002.

RATIONAL. **Rational Rose**, Disponível em: <http://www.rational.com>. Acesso em: 05 jul. 2002.

SEI – SOFTWARE ENGINEERING INSTITUT. **A Framework for Software Product Line Practice - Version 3.0**. Disponível em: <http://www.sei.cmu.edu/plp/frame_report/productDA.htm>, Acesso em: 05 jul. 2002.

SPC - Software Productivity Consortium. **Reuse-Driven Software Processes Guidebook.** SPC-92019-CMC version 02.00.03 November 1993.

SUN. **The Source for Java Technology**. Disponível em: <http://www.java.sun.com>, Acesso em: 25 jun. 2002.

TANAKA, S. **Um Framework de Agenda de Tarefas para Gerenciadores de Processos**. 2000. 124p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

WARMER, J.; KLEPPE, A. **The Object Constraint Language, Precise Modeling with UML**.1. ed. EUA: Addison-Wesley Publishing Company, 1998.

WEISS, D. M.; CHI TAU, R. L. ***Software Product-Line Engineering: A Family-Based Software Development Approach.*** Addison-Wesley, 1999.

WERNER, C. M. L., BRAGA, R. M. M. **Desenvolvimento Baseado em Componentes.** Proc. of the nd XIV SBES, 2000, João Pessoas. João Pessoa: SBC, 2000. p. 297-329.

WfMC - WORKFLOW MANAGEMENT COALITION. **Workflow: An Introduction**. Disponível em: http://www.wfmc.org/standards/docs/*Workflow*_An_Introduc-tion.pdf. Acesso em: 25 ago. 2001.

WfMC - WORKFLOW MANAGEMENT COALITION. ***Workflow* Reference Model**. Document number TC00-1003, January 19, 1995. 55 p.

WFMC - WORKFLOW MANAGEMENT COALITION. **Terminology & Glossary**. Bruxelas, Jun. 1999. 52p.

ANEXO 1

Descrição das classes e tabelas mapeadas para o ObjectBridge.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE MappingRepository SYSTEM "repository.dtd">

<MappingRepository>
<!-- The Default JDBC Connection. If a Class does not specify its own JDBC
Connection,
  the Connection specified here will be used. -->

 <JdbcConnectionDescriptor id="default">
  <dbms.name>MySQL</dbms.name>
  <driver.name>org.gjt.mm.mysql.Driver</driver.name>
  <url.protocol>jdbc</url.protocol>
  <url.subprotocol>mysql</url.subprotocol>
  <url.dbalias>//200.111.111.203/WfMS</url.dbalias>
  <user.name>rdms</user.name>
  <user.passwd>123</user.passwd>
 </JdbcConnectionDescriptor>

<!-- Definitions for User -->
 <ClassDescriptor id="1">
  <class.name>bSystemSpecification.aStaticModel.User</class.name>
  <table.name>User</table.name>

  <FieldDescriptor id="1">
   <field.name>userId</field.name>
   <column.name>userId</column.name>
   <jdbc_type>INTEGER</jdbc_type>
   <PrimaryKey>true</PrimaryKey>
  </FieldDescriptor>

  <FieldDescriptor id="2">
   <field.name>userName</field.name>
   <column.name>userName</column.name>
   <jdbc_type>VARCHAR</jdbc_type>

  </FieldDescriptor>


  <FieldDescriptor id="3">
   <field.name>userPassword</field.name>
```

```
  <column.name>userPassword</column.name>
  <jdbc_type>VARCHAR</jdbc_type>
 </FieldDescriptor>
</ClassDescriptor>

<!-- Definitions for Task -->
<ClassDescriptor id="2">
 <class.name>bSystemSpecification.aStaticModel.Task</class.name>
 <table.name>Task</table.name>

 <FieldDescriptor id="1">
  <field.name>workflowId</field.name>
  <column.name>workflowid</column.name>
  <jdbc_type>INTEGER</jdbc_type>
  <PrimaryKey>true</PrimaryKey>
 </FieldDescriptor>

 <FieldDescriptor id="2">
  <field.name>userId</field.name>
  <column.name>userId</column.name>
  <jdbc_type>INTEGER</jdbc_type>
  <PrimaryKey>true</PrimaryKey>
 </FieldDescriptor>

 <FieldDescriptor id="3">
  <field.name>taskId</field.name>
  <column.name>taskId</column.name>
  <jdbc_type>INTEGER</jdbc_type>
  <PrimaryKey>true</PrimaryKey>
 </FieldDescriptor>

 <FieldDescriptor id="4">
  <field.name>taskImplementation</field.name>
  <column.name>taskImplementation</column.name>
  <jdbc_type>VARCHAR</jdbc_type>
 </FieldDescriptor>

 <FieldDescriptor id="5">
  <field.name>taskDescription</field.name>
  <column.name>taskDescription</column.name>
  <jdbc_type>VARCHAR</jdbc_type>
 </FieldDescriptor>

 <FieldDescriptor id="6">
  <field.name>taskCharacteristic</field.name>
  <column.name>taskCharacteristic</column.name>
  <jdbc_type>VARCHAR</jdbc_type>
 </FieldDescriptor>
```

```
<FieldDescriptor id="7">
 <field.name>taskInstantiation</field.name>
 <column.name>taskInstantiation</column.name>
 <jdbc_type>VARCHAR</jdbc_type>
</FieldDescriptor>

    <FieldDescriptor id="8">
 <field.name>taskStartMode</field.name>
 <column.name>taskStartMode</column.name>
 <jdbc_type>VARCHAR</jdbc_type>
</FieldDescriptor>

    <FieldDescriptor id="9">
 <field.name>taskFinishMode</field.name>
 <column.name>taskFinishMode</column.name>
 <jdbc_type>VARCHAR</jdbc_type>
</FieldDescriptor>

    <FieldDescriptor id="10">
 <field.name>taskDuraction</field.name>
 <column.name>taskDuraction</column.name>
 <jdbc_type>INTEGER</jdbc_type>
</FieldDescriptor>

    <FieldDescriptor id="11">
 <field.name>taskPreCondition</field.name>
 <column.name>taskPreCondition</column.name>
 <jdbc_type>VARCHAR</jdbc_type>
</FieldDescriptor>

    <FieldDescriptor id="12">
 <field.name>taskPostCondition</field.name>
 <column.name>taskPostCondition</column.name>
 <jdbc_type>VARCHAR</jdbc_type>
</FieldDescriptor>

    <FieldDescriptor id="13">
 <field.name>taskStatus</field.name>
 <column.name>taskStatus</column.name>
 <jdbc_type>VARCHAR</jdbc_type>
</FieldDescriptor>
</ClassDescriptor>

<!-- Definitions for Workflow -->
 <ClassDescriptor id="3">
  <class.name>bSystemSpecification.aStaticModel.Workflow</class.name>
  <table.name>Workflow</table.name>

  <FieldDescriptor id="1">
```

```
    <field.name>workflowId</field.name>
    <column.name>workflowId</column.name>
    <jdbc_type>INTEGER</jdbc_type>
    <PrimaryKey>true</PrimaryKey>
   </FieldDescriptor>

   <FieldDescriptor id="2">
    <field.name>workflowVersion</field.name>
    <column.name>workflowVersion</column.name>
    <jdbc_type>VARCHAR</jdbc_type>

   </FieldDescriptor>

   <FieldDescriptor id="3">
    <field.name>workflowVendor</field.name>
    <column.name>workflowVendor</column.name>
    <jdbc_type>VARCHAR</jdbc_type>
   </FieldDescriptor>


   <FieldDescriptor id="4">
    <field.name>workflowCreated</field.name>
    <column.name>workflowCreated</column.name>
    <jdbc_type>VARCHAR</jdbc_type>
   </FieldDescriptor>

   <FieldDescriptor id="5">
    <field.name>workflowDescription</field.name>
    <column.name>workflowDescription</column.name>
    <jdbc_type>VARCHAR</jdbc_type>
   </FieldDescriptor>

   <FieldDescriptor id="6">
    <field.name>workflowDuraction</field.name>
    <column.name>workflowDuraction</column.name>
    <jdbc_type>INTEGER</jdbc_type>
   </FieldDescriptor>

  </ClassDescriptor>

<!-- Definitions for Precondition-->
  <ClassDescriptor id="4">

<class.name>cArchitecturalDesign.Layers.bApplicationLayer.bExecuteWorkflowMgr.
PreConditionMgr</class.name>
   <table.name>PreCondition</table.name>
   <FieldDescriptor id="1">
    <field.name>workflowId</field.name>
    <column.name>workflowId</column.name>
```

```
    <jdbc_type>INTEGER</jdbc_type>
    <PrimaryKey>true</PrimaryKey>
   </FieldDescriptor>

  <FieldDescriptor id="2">
    <field.name>userId</field.name>
    <column.name>userId</column.name>
    <jdbc_type>INTEGER</jdbc_type>
    <PrimaryKey>true</PrimaryKey>
  </FieldDescriptor>

  <FieldDescriptor id="3">
    <field.name>taskId</field.name>
    <column.name>taskId</column.name>
    <jdbc_type>INTEGER</jdbc_type>
    <PrimaryKey>true</PrimaryKey>
  </FieldDescriptor>

 </ClassDescriptor>

<!-- Definitions for TimerMgr-->
 <ClassDescriptor id="5">

<class.name>cArchitecturalDesign.Layers.bApplicationLayer.bExecuteWorkflowMgr.
TimerMgr</class.name>
  <table.name>Timer</table.name>
  <FieldDescriptor id="1">
    <field.name>workflowId</field.name>
    <column.name>workflowId</column.name>
    <jdbc_type>INTEGER</jdbc_type>
    <PrimaryKey>true</PrimaryKey>
   </FieldDescriptor>

  <FieldDescriptor id="2">
    <field.name>userId</field.name>
    <column.name>userId</column.name>
    <jdbc_type>INTEGER</jdbc_type>
    <PrimaryKey>true</PrimaryKey>
  </FieldDescriptor>

  <FieldDescriptor id="3">
    <field.name>taskId</field.name>
    <column.name>taskId</column.name>
    <jdbc_type>INTEGER</jdbc_type>
    <PrimaryKey>true</PrimaryKey>
  </FieldDescriptor>

  <FieldDescriptor id="4">
    <field.name>startAt</field.name>
```

```
    <column.name>startAt</column.name>
    <jdbc_type>VARCHAR</jdbc_type>
    <PrimaryKey>false</PrimaryKey>
   </FieldDescriptor>

   <FieldDescriptor id="5">
    <field.name>finishAt</field.name>
    <column.name>finishAt</column.name>
    <jdbc_type>VARCHAR</jdbc_type>
    <PrimaryKey>false</PrimaryKey>
   </FieldDescriptor>
  </ClassDescriptor>

<!-- Definitions for TaskTransition-->
  <ClassDescriptor id="6">

<class.name>cArchitecturalDesign.Layers.bApplicationLayer.bExecuteWorkflowMgr.
TaskTransition</class.name>
   <table.name>TaskTransition</table.name>
   <FieldDescriptor id="1">
    <field.name>workflowId</field.name>
    <column.name>workflowId</column.name>
    <jdbc_type>INTEGER</jdbc_type>
    <PrimaryKey>true</PrimaryKey>
   </FieldDescriptor>

   <FieldDescriptor id="2">
    <field.name>userId</field.name>
    <column.name>userId</column.name>
    <jdbc_type>INTEGER</jdbc_type>
    <PrimaryKey>true</PrimaryKey>
   </FieldDescriptor>

   <FieldDescriptor id="3">
    <field.name>taskId</field.name>
    <column.name>taskId</column.name>
    <jdbc_type>INTEGER</jdbc_type>
    <PrimaryKey>true</PrimaryKey>
   </FieldDescriptor>

   <FieldDescriptor id="4">
    <field.name>transitionId</field.name>
    <column.name>transitionId</column.name>
    <jdbc_type>INTEGER</jdbc_type>
    <PrimaryKey>true</PrimaryKey>
   </FieldDescriptor>

   <FieldDescriptor id="5">
    <field.name>transitionFrom</field.name>
```

```
    <column.name>transitionFrom</column.name>
    <jdbc_type>INTEGER</jdbc_type>
    <PrimaryKey>false</PrimaryKey>
   </FieldDescriptor>

   <FieldDescriptor id="6">
    <field.name>transitionTo</field.name>
    <column.name>transitionTo</column.name>
    <jdbc_type>INTEGER</jdbc_type>
    <PrimaryKey>false</PrimaryKey>
   </FieldDescriptor>

   <FieldDescriptor id="7">
    <field.name>transitionDescription</field.name>
    <column.name>transitionDescription</column.name>
    <jdbc_type>VARCHAR</jdbc_type>
    <PrimaryKey>false</PrimaryKey>
   </FieldDescriptor>

   <FieldDescriptor id="8">
    <field.name>transitionDuraction</field.name>
    <column.name>transitionDuraction</column.name>
    <jdbc_type>INTEGER</jdbc_type>
    <PrimaryKey>false</PrimaryKey>
   </FieldDescriptor>

  </ClassDescriptor>


  <!-- END OF OJB INTERNAL MAPPINGS-->
</MappingRepository>
```

ANEXO 2

Comandos SQL para criação das tabelas utilizadas na implementação do protótipo. O banco de dados relacional utilizado foi o MySQL.

Entidade: PreCondition

```
CREATE TABLE `PreCondition` (
  `workflowId` int(11) NOT NULL default '0',
  `userId` int(11) NOT NULL default '0',
  `taskId` int(11) NOT NULL default '0',
  PRIMARY KEY  (`workflowId`,`userId`,`taskId`)
) TYPE=MyISAM;
```

Entidade:Task

```
CREATE TABLE `Task` (
  `workflowId` int(11) NOT NULL default '0',
  `userId` int(11) NOT NULL default '0',
  `taskId` int(11) NOT NULL default '0',
  `taskImplementation` varchar(50) default NULL,
  `taskDescription` varchar(50) default NULL,
  `taskCharacteristic` varchar(50) default NULL,
  `taskInstantiation` varchar(50) default NULL,
  `taskStartMode` varchar(50) default NULL,
  `taskFinishMode` varchar(50) default NULL,
  `taskDuraction` int(11) default NULL,
  `taskPreCondition` varchar(50) default NULL,
  `taskPostCondition` varchar(50) default NULL,
  `taskStatus` varchar(10) default NULL,
  PRIMARY KEY  (`workflowId`,`userId`,`taskId`)
) TYPE=MyISAM;
```

Entidade: TaskTransition

```
CREATE TABLE `TaskTransition` (
  `workflowId` int(11) NOT NULL default '0',
  `userId` int(11) NOT NULL default '0',
  `taskId` int(11) NOT NULL default '0',
  `transitionId` int(11) NOT NULL default '0',
  `transitionFrom` int(11) NOT NULL default '0',
  `transitionTo` int(11) NOT NULL default '0',
  `transitionDescription` varchar(50) default NULL,
  `transitionDuraction` int(11) NOT NULL default '0',
  PRIMARY KEY  (`workflowId`,`userId`,`taskId`,`transitionId`)
) TYPE=MyISAM;
```

Entidade: Timer

```
CREATE TABLE `Timer` (
  `workflowId` int(11) NOT NULL default '0',
  `userId` int(11) NOT NULL default '0',
  `taskId` int(11) NOT NULL default '0',
  `startAt` varchar(8) NOT NULL default '',
  `finishAt` varchar(8) NOT NULL default '',
  PRIMARY KEY  (`workflowId`,`userId`,`taskId`)
) TYPE=MyISAM;
```

Entidade: User

```
CREATE TABLE `User` (
  `userId` int(11) NOT NULL default '0',
  `userName` varchar(30) NOT NULL default '',
  `userPassword` varchar(18) default NULL,
  PRIMARY KEY  (`userId`)
) TYPE=MyISAM;
```

Entidade: Workflow

```
CREATE TABLE `Workflow` (
  `workflowId` int(11) NOT NULL default '0',
  `workflowVersion` varchar(50) default NULL,
  `workflowVendor` varchar(30) default NULL,
  `workflowCreated` varchar(10) default NULL,
  `workflowDescription` varchar(100) default NULL,
  `workflowDuraction` int(11) default NULL,
  PRIMARY KEY  (`workflowId`)
) TYPE=MyISAM;
```